5-2011

Strengthening the Anonymity of Anonymous Communication Systems

Christopher Abbott
Clemson University, cdabbott316@gmail.com

Strengthening the Anonymity of Anonymous Communication Systems

---

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

---

by
Christopher D. Abbott
May 2011

Accepted by:
Dr. Richard R. Brooks, Committee Chair
Dr. Kuang-Ching Wang
Dr. Robert J. Schalkoff

Abstract

In this work, we examine why a popular anonymity network, Tor, is vulnerable to timing side-channel attacks. We explore removing this vulnerability from Tor without sacrificing its low-latency which is important for usability.

We find that Tor is vulnerable because inter-packet delays propagate along the network path from the source to the destination. This provides an easily detected signature. We explore techniques for making the timing signature either expensive or impossible to detect.

If each packet took a unique, disjoint path from source to destination the inter-packet delay signature would be undetectable. Jitter and latency would change packet arrival orders. This is impractical since the overhead for constructing these circuits would be prohibitive. We scaled this idea back to reflect how the BitTorrent protocol creates a large number of possible paths from a small number of nodes.

We form a fully connected network with the source, destination, and a small number of nodes. The number of paths through this network from source to destination grows quickly with the addition of each node. Paths do not have to include every node, so the delay of each path is different. By transmitting consecutive packets on different paths, the network delays will mask the inter-packet delay signature.

Dedication

To my family, for being there always.

Acknowledgments

Foremost, I would like to thank my advisor, Dr. Richard R. Brooks. Without your guidance and assistance my work would not have been possible. Our conversations have always been thought-provoking and your recommendations well-intentioned. You have imparted a lasting impression of how to approach the challenging issues present in computer security.

I would also like to thank Dr. Kuang-Ching Wang and Dr. Robert J. Schalkoff for serving on my committee and for your interesting courses. While I have not used all the concepts presented in your classes in this work, they have proved invaluable in looking at computer security problems from a different angle.

I have also had the privilege of working alongside and after a talented group of students in our research group. My work would not have been possible without the previous work done by Jason Schwier and Ryan Craven. Being able to converse and exchange ideas with the current research group has been an enriching experience and assisted me more that I thought possible.

Finally, I would like to thank the Holcombe Department of Electrical and Computer Engineering. The monetary assistance from my teaching assistantship was a blessing, and the staff and faculty were excellent. I will remember my interactions with them forever.

This material is based on research sponsored by the Air Force Research Labora-

tory, under agreement number AFD-080228-008. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computer and network security typically refers to the confidentiality, integrity, availability and assurance of resources and data. These topics have been and continue to be widely investigated to ensure computers and networks provide acceptable levels of security to their users. While these properties are important to the user, they do nothing to provide or ensure the user's anonymity.

There have been many diverse strategies throughout history for covert communication. These techniques largely involve modifying the transport layer to physically hide the communication from potential threats. However, we are unable to modify the transport layer and instead are forced to use some form of mix network. Mix networks were first proposed by Chaum and are networks that use secure proxy chains to transmit data [10].

Applications for these types of networks exist in both the military and civilian sectors. The military utilizes these networks to hide the originating and receiving party's identity and location from their adversaries. Civilians utilize them to better protect their privacy when publishing or transmitting sensitive information, such as health, tax, or banking records. These networks can also help to better protect

organizations and users from persecution and circumvent censorship tools such as the "great firewall" [16].

In both of these cases, there exist powerful and motivated adversaries that wish to determine the identity of the various users. It is the subject of this thesis to study and improve anonymous communication networks, with the goal of providing stronger anonymity to their users.

1.1 Anonymous Communication

Current anonymous networks provide users with the ability to communicate while hiding their identity and location from other parties. The anonymity they provide does not ensure privacy though. Users must undertake additional measures to ensure their communication is private.

Many of the anonymous networks use some form of proxying to provide users anonymity. Instead of directly sending traffic from one user to another, the networks bounce the traffic from one user through some set of nodes before sending it to the other user. This prevents the header information of the traffic from containing both the source and the destination. Depending on which network is used, it is possible that no individual node knows both the source and destination. These networks can also employ other techniques to protect user anonymity, such as:

- Padding packets to a fixed length
- Mixing packet flows
- Batching or controlling packet flow rates
- Sending dummy traffic

Depending on how the networks implement different combinations of these options, they are classified as either low-latency or high-latency. The high-latency networks offer better protection against traffic analysis[1]. These networks generally have strong mixing, batching, and reordering algorithms that eliminate information used in traffic analysis at the cost of adding large delays to individual packets [27]. This extra delay makes the high-latency networks a poor choice for interactive communication. On the other hand, the low-latency networks which can support interactive communication implement fewer of these additional techniques. As a result, they provide less protection against traffic analysis.

High-latency networks are typically called remailers. Mixmaster [28] and Mixminion [15] are two examples of this type of network. These are well suited for sending anonymous email or posting anonymous messages to message boards or organizations. The low-latency networks are typically called anonymous overlay networks or Onion Routing (OR) networks. The Onion Router (Tor) [18], Invisible Internet Project (I2P) [23], and Java Anon Proxy (JAP or JonDonym) [6] are examples of this type of network. These are well suited for interactive communication such as SSH, HTTP, or various P2P protocols.

1.2 Traffic Analysis

Attackers are often unable to view unencrypted digital communication. The use of encryption or code words has become more popular outside of the military sector as encryption algorithms have become better understood, better implemented and easier to use [14]. As a result, attackers have to use other methods to determine information about their adversary's communication.

[1]Traffic analysis uses patterns found in intercepted communication to determine information about the communicators.

Traffic analysis is one method attackers can use to do this. The roots of traffic analysis can be traced back to World War One and has had an impact in many military conflicts since [26]. Consequently, the military has devoted considerable resources to improving their traffic analysis methods and securing their communication methods against them. The growth in popularity of personal computers and the internet made the use of traffic analysis possible in the civilian sector.

Even though the attacker cannot directly see the communication between users, the attacker can still possibly determine various facts about traffic:

- Packet headers
- Packet lengths
- Timing between packets

The attacker can collect these bits of information. By themselves, the bits may tell the attacker very little. If the attacker is able to piece all of the bits together from a single traffic flow, then the attacker can determine larger pieces of information about the users [19], such as:

- Who is talking to whom
- Where the users are located
- When the users communicate
- What applications or protocols are being used
- How much data is being communicated

When the user is communicating sensitive information, keeping these details private is a larger issue. The attacker determining any of them could be grounds to compromise the user.

1.3 Research Question

Recent advances in pattern recognition [34, 35] have made more sophisticated traffic analysis and side-channel attacks possible [7, 13]. These new attacks represent new threats to digital communication and propose unique challenges to the users of anonymous communication networks. We ask, what can be done to help strengthen user anonymity in these networks?

We focus our research on Tor. Recent work shows that the threat to Tor's anonymity is real [13]. This attack matches a session's source to its sink by the inter-packet delay signature in the communication protocol. We consider the underlying vulnerability to be using point-to-point connections for anonymous sessions.

We propose having sessions interleave multiple paths to modify this signature and maintain Tor's low-latency, which is important for usability. This idea is inspired from the BitTorrent protocol creating a large number of paths among a small number of nodes. Each path will have a different latency and jitter. By using different paths to transmit consecutive packets, the network will mask the inter-packet delay signature of the protocol and make it more difficult or impossible to detect.

Now, our question is can we model this idea to better understand it? Also, can we implement our design and measure its effect on the inter-packet delay signature?

1.4 Organization

Previous sections in Chapter 1 outline the motivation behind our work. Users require stronger protection as traffic analysis has grown more powerful. We present our research question of how we can better protect anonymous communication networks against traffic analysis.

Chapter 2 explains background material. We look at what anonymity is and how Tor provides anonymity. We examine attacks on Tor and why Tor is vulnerable to attack. OnionCat is discussed since it allows a wide range of applications to use anonymous communication networks. Side-channel attacks are examined, particularly the Tor side-channel attack we hope to disable.

Chapter 3 describes our method for protecting Tor from side-channel attacks. We develop a mathematical model to represent our method and protocols it should protect. The effect of adding additional nodes is quantified and the number of paths a set of nodes form is calculated. We also examine how the varying network delays will affect the inter-packet delay signature and the effect this will have on the CSSR[2] algorithm.

Chapter 4 describes our channel implementation and the experimental results. We collect data to justify several of our assumptions in developing the mathematical model. The signatures at the source and receiver ends of our channel are captured using different number of nodes to form the paths. This data is used by the side-channel attack to demonstrate the effect the channel has on the signature.

Chapter 5 contains a summary of our results and the conclusions we were able to draw from them. We also suggest interesting ideas and questions to continue investigating.

[2]Causal State Splitting and Reconstruction

Chapter 2

Background

Tor is a distributed, anonymous communications network, which focuses on providing users with anonymous, private web browsing [2]. Because the network is fast enough for web browsing, it is vulnerable to sophisticated timing attacks [43]. The focus of our research is to design and build a network within Tor that is both fast enough for general web browsing and eliminates the vulnerabilities from these timing attacks.

2.1 Anonymity

In everyday use, the word anonymous has several meanings but commonly refers to an object of unknown origin[1]. Converting this definition into technical terms for digital communication is not a trivial task but is one undertaken in [33], where anonymity is divided into three parts – sender anonymity, recipient anonymity and unlinkability of sender and receiver.

2.1.1 Anonymity Set

One way to determine sender and recipient anonymity is through the use of the anonymity set first proposed in [10]. For sender anonymity, the anonymity set is the subset of participants that may have originally sent the message. For receiver anonymity, the anonymity set is the subset of participants that may be the final destination of the message. Both of these are reasonable metrics for logicians and cryptographers.

While the anonymity set allows us to qualify anonymity, it does not make it easy to quantify the results. [36] has proposed the use of the entropy function to help quantify the meaning of anonymity sets. By using the distribution entropy:

$$S = -\sum_{u \in \Psi} p_u \log_2(p_u) \tag{2.1}$$

where Ψ is the set of users and p_u is the probability user u originally sent the message. Using this metric, larger values of S mean the sender has more anonymity.

2.1.2 Unlinkability

The term unlinkability means that *a posteriori* knowledge gained from observing the message pass through the system does not change the probability that the sender and recipient can be linked together [25]. One important distinction to make is this does not mean the message passing through the system is unobservable, simply that there are no observable qualities in the message that link sender to recipient.

This gives us another way to qualify anonymity in any system. For anonymity to exist, both the sender and recipient must remain unlinkable. So for each message passing through the system, the message must not be linked to both the sender and

the recipient. The anonymity set is one way to measure the link between the message and the sender. It also tells us that the act of sending the message through the system must not link the sender and recipient together.

This last item is one that many side channel attacks focus on exploiting. Consequently, it is also one we will focus on strengthening in our research.

2.2 Tor Anonymity Assumptions (Attack Models)

The Tor network is a distributed, anonymous overlay network. It provides users with anonymous, private communication channels usable for a variety of activities. Privacy for user data is easily obtained by using cryptography. This makes the data almost impossible to read without the decryption key, but unfortunately does nothing to make the transaction anonymous. Even though data is encrypted, the routing information is still available in cleartext. From this information, attackers can use traffic analysis to determine various facts about the session, such as the sender, receiver, size and so forth.

Tor provides anonymity by ensuring both the sender and receiver cannot be determined from routing information. It does this by creating a tunnel through the network for the data packets. When the sender wants to send a data packet to some user, Tor encapsulates the packet within another packet for each node in the tunnel. By doing this, no node within the tunnel knows who the sender and the receiver are. If the receiver tries to send a packet back, the packet will travel the opposite way through the tunnel and then be sent to the sender.

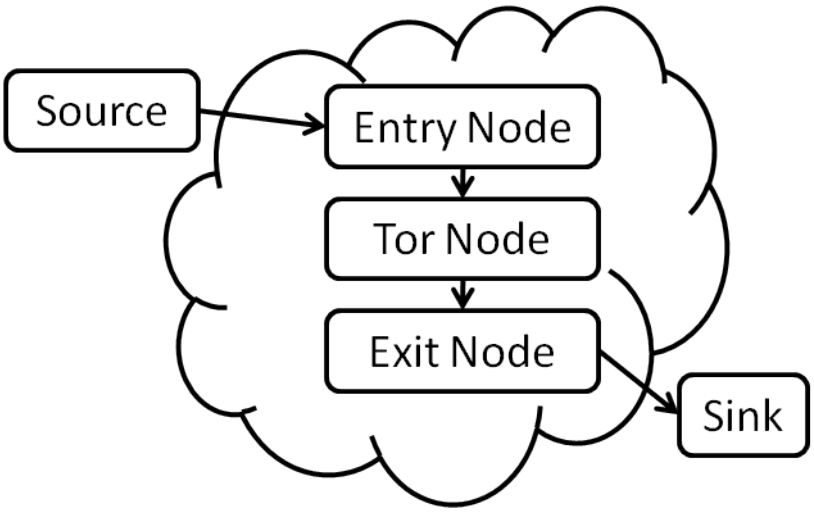

Figure 2.1: Example Tor circuit

2.2.1 Construction of a Tor Circuit

The sender's Tor client is responsible for constructing the circuit. It does this with the aid of a directory server, which contains a list of all the Tor routers currently operating and what type of connections they will support. The sender begins by picking three routers to build its circuit, and begins the circuit with the first router. This step involves exchanging keys with the first router and setting up an encrypted tunnel between the two nodes (sender and router 1).

Once the first tunnel has been established, the sender tells the first router to extend the circuit to the second router. The first router exchanges keys with the second and sets up an encrypted tunnel between the two routers. Once this tunnel is established, the first router tells the sender it was successful and the key for the second router.

The sender then tells the second router to extend the circuit to the third router. It does this by sending a command through the partially built circuit. The second and third routers then exchange keys. The confirmation and key are sent back

up the circuit to the sender. Once the sender has confirmed the three router circuit it can begin to use the circuit to send data to some receiver. Traffic is sent down the channel by encapsulating packets and having each router use its key to remove a layer of encryption before sending the packet further down the circuit. An illustrated example is now given of this process.

2.2.2 Sending and Receiving Data in Tor

After the circuit is constructed, the sender (Alice) knows the keys for each router in the circuit (TorNode1, TorNode2, and TorNode3). Alice can also establish a key with the receiver (Bob), and use it to encrypt their communication. It is important to note that Tor provides anonymity, not privacy. If Alice sends unencrypted messages to Bob, the last router in the circuit and anyone spying on the connection after that point will be able to read the message.

To send a message to Bob, Alice begins by constructing a packet with her message from TorNode3 to Bob. She then encrypts this packet with the key for TorNode3 and uses this as the data for a packet from TorNode2 to TorNode3. She then encrypts this packet with the key for TorNode2 and uses this as the data for a packet from TorNode1 to TorNode2. Finally, she encrypts this packet with the key for TorNode1 and uses this as the data for a packet from Alice to TorNode1.

Alice sends the packet to TorNode1, who uses its key with Alice to decrypt the packet and finds a packet that it needs to send to TorNode2. TorNode2 receives this packet from TorNode1 and decrypts it with its key and finds a packet it needs to send to TorNode3. TorNode3 receives this packet from TorNode2 and decrypts it with its key and finds a packet it needs to send to Bob. Bob then receives the packet from TorNode3, and has no idea Alice sent it unless Alice has told him so in the message.

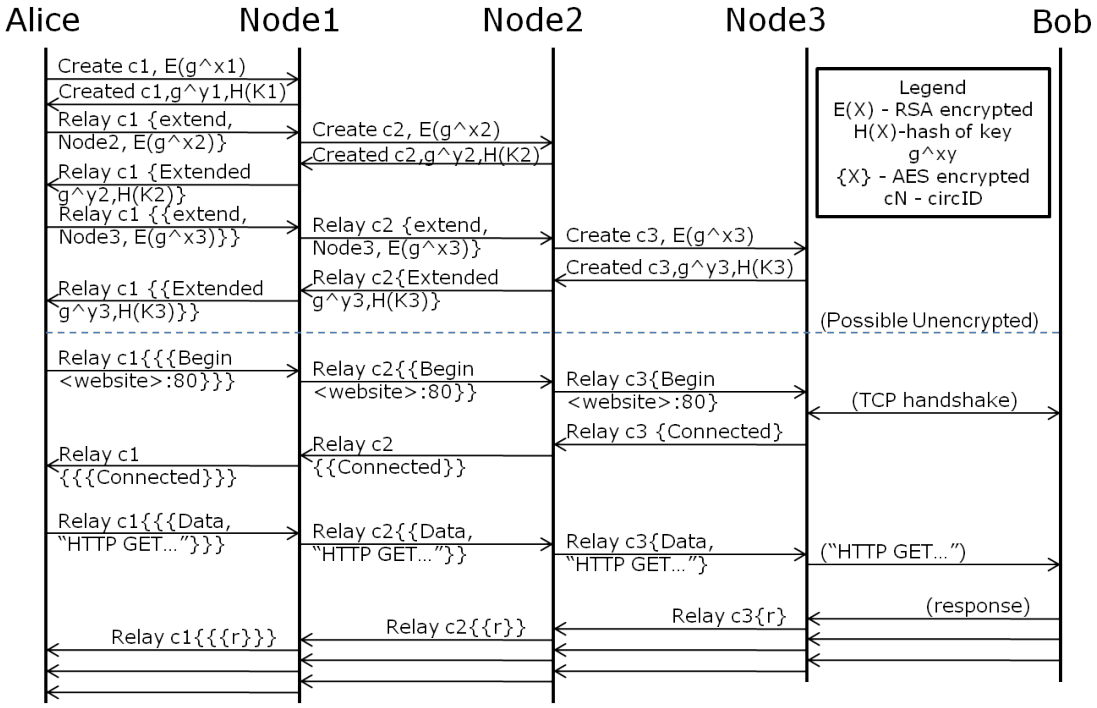

Figure 2.2: Example of two users communicating in Tor

If Bob wants to send a packet back to Alice, he has to send it back through the Tor circuit. Let's say Bob is a web server and Alice has requested some web page she wants to access anonymously. Bob doesn't know it was Alice who requested the page; he received the request from TorNode3. Bob replies to TorNode3 who encrypts the reply with its key and sends the encrypted response up the circuit to TorNode2. TorNode2 receives the packet from TorNode3, encrypts it with its key and sends it up the circuit to TorNode1. TorNode1 receives the packet from TorNode1, encrypts it with its key and sends it up the circuit to Alice. Alice receives this packet and then uses the three keys to decrypt it, and finds the web page she requested from Bob.

2.2.3 Assumptions and Vulnerabilities

Tor's solution to anonymity is vulnerable to the exit node in the tunnel being compromised by the attacker, meaning the attacker controls the node or can watch all traffic passing through the node. If the attacker has compromised this node, then the attack knows who the destination is and can possibly see the message in cleartext. Then the attacker can try to collude with other nodes in the tunnel, or try to correlate the amount and timing of the exit traffic with the traffic at some potential source.

Other nodes in the tunnel are less vulnerable because breaking the encryption on the message is a difficult problem. The attacker is unlikely to have the resources to complete this step in a reasonable amount of time. Since this process will take a long time, it is unlikely the user's anonymity will matter at that point.

It is also difficult for the attacker to collude with the other nodes in the tunnel. The attacker will at least need to collude with the first and last node in the tunnel to determine the sender and receiver. Let's say an attacker can control c nodes out of the n nodes in the Tor network. Then the attacker will have a $(c/n)^2$ chance

of controlling the first and last nodes in the tunnel. As Tor continues to grow its user base, this probability will only drop. Various groups have looked into ways of increasing the size of c, and have limited success with this approach. Tor enforces that each node in the tunnel must be on a separate subnet, so one user with several thousand hosts or virtual machines will fail. Another group tried to use botnets but even though they controlled a large part of the network, the botnet's bandwidth was not enough to control a large enough portion of the traffic [30].

The last attempt is the most likely to occur within Tor. As in the previous case, the attacker has a c/n chance of controlling the first node in the tunnel and the attacker can monitor the website the user is connecting to. If Tor selects a new entry node for the tunnel for each session, then the attacker will eventually see one of the sessions. Breaking the user's anonymity once can have the same consequences as breaking it every time, so this is a big issue. Tor's solution to this problem is to create entry guards, or a set of nodes the user selects at random or by some trust algorithm to be used as entry points to their tunnels. By fixing the set of entry nodes to c nodes, the attacker's probability of controlling an entry node is c/n. As n increases and c remains constant, the user's probability of avoiding the attacker's attacks increases [32].

2.3 OnionCat

OnionCat creates a VPN within Tor (or I2P) for a closed group of users [21]. OnionCat assigns an IPv6 address to this network making it easy to send any IP based traffic through the network. This allows us to use any applications that support IPv6 within the Tor network. It also connects us to every user in our group through the VPN it creates. Alternatively, it could create an open anonymous network for us, but

our proposed network will have a fixed group of users.

Just like any other VPN, OnionCat encapsulates IP packets within IP packets. One difference from other VPN's is OnionCat uses Tor to transmit its data rather than some data link layer such as Ethernet. Since OnionCat's virtual circuits are contained in Tor, our traffic will not have to exit Tor and travel through Tor's high traffic exit nodes.

Two users connecting to each other using OnionCat resembles a client connecting to a server. One user will be the client and initialize the connection to the other who accepts it. In Tor, the server is a hidden service, meaning both the server and client cannot determine the identity and location of the other from the network connection.

OnionCat allows each user to be both client and server by requiring each user to set up a hidden service in Tor. This allows any OnionCat user to connect to any other OnionCat user, provided the user knows the other's onion-URL. An onion-URL is Tor's address for a user's hidden service and is derived from the user's private key. OnionCat forms the user's IPv6 address with a 48-bit prefix and an 80-bit address derived from the user's onion-URL [21]. By using this scheme, it is easy for OnionCat to translate one address into the other and route traffic into or out of Tor.

2.3.1 Construction of an OnionCat Circuit

OnionCat uses Tor's hidden services to construct its circuits. Hidden services in Tor are a way for a user to offer some service anonymously. We will adapt the previous example of Bob hosting some web page Alice wants to access.

Bob first establishes his web page as a hidden service in Tor. He does this by creating circuits with several Tor nodes that will be used as introduction points. This

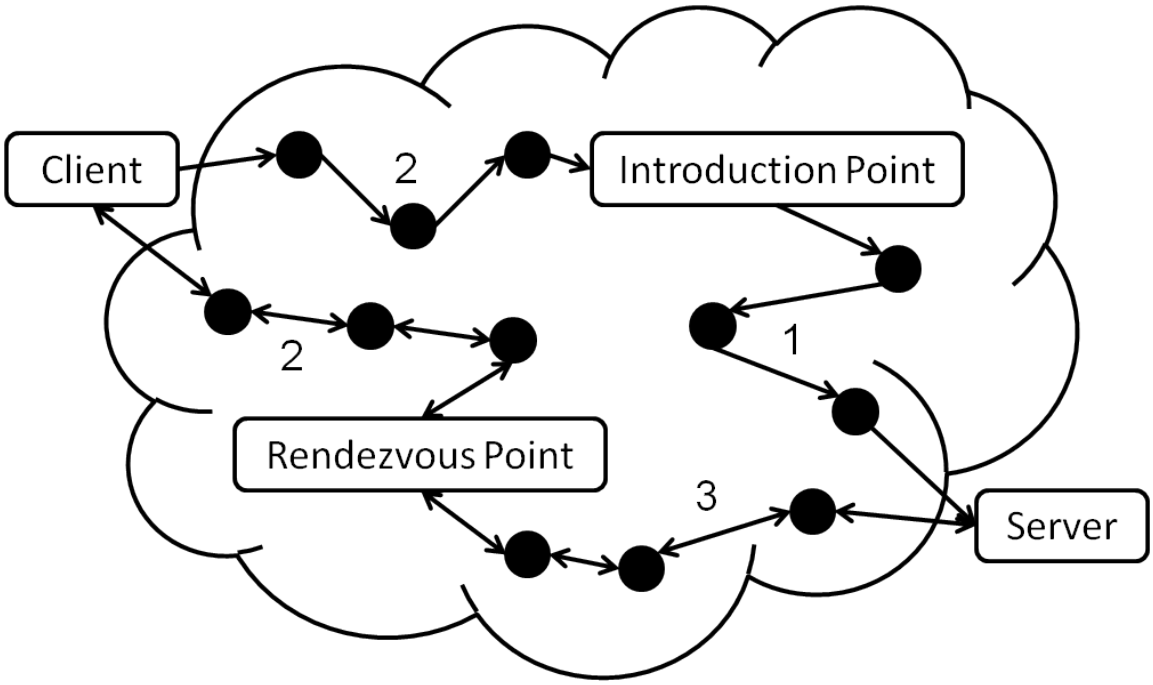

Figure 2.3: Example of a circuit in OnionCat

is Tor's name for the node that will serve as the initial contact node between Alice and Bob. Bob then registers his hidden service, his introduction points, and a public key with a distributed hash table among all of Tor's routers running for more than a day. Bob needs to sign his registration with his private key to prove he is the owner of the hidden service.

Alice can now request Bob's hidden service with its onion-url from the distributed hash table. Alice also constructs a Tor circuit with one Tor node that will be used as a rendezvous point by giving it a one-time secret key. This is Tor's name for the node that will connect Alice's circuit with it to Bob's circuit with it. Alice then sends a request for Bob's hidden service through one of the introduction points with her rendezvous point and her one time secret key.

Bob receives Alice's request from the introduction point and creates his own circuit to the rendezvous point. By supplying the rendezvous point with Alice's one

time secret key, the rendezvous point knows to bridge the two circuits together. Alice can now access Bob's hidden service and Bob can service her requests while neither knows who or where the other is.

2.4 Side Channel Attacks

A side channel attack refers to any type of attack which uses knowledge gained from observations of the system to compromise it [7]. These types of attacks have been used to compromise both communication networks and cryptographic algorithms, and have grown more popular as cryptography algorithms have become more robust and theoretically sound.

A variety of side channel attacks have been developed to determine or reduce the set of cryptographic keys used in RSA. These attacks depend on information gained from observing timing information [8], power consumption [24] or electromagnetic radiation leaks. While these types of attacks could be used to break encryption in anonymous network to break anonymity, side channel attacks on the anonymous network have been equally effective.

SSH is vulnerable to side channel attacks based on the delays between packets corresponding to the delays between user key strokes. Song et al realized that these delays contained information about the pair of keys and developed a method using hidden Markov models to predict which key pairs generated the delay [41]. Using this model, they were effectively able to reduce the search space for passwords entered in interactive SSH sessions and greatly reduce the time required to find the password.

Similarly, Hintz recognized that individual websites contain identifying characteristics from their resources, such as the sizes of their HTML, CSS, and images. By creating a profile of websites, it became possible to discover which website a user

visits even if the site is accessed through HTTPS [22]. This approach has also been modified to use the size of the search suggestions to determine what a user is typing into a search bar [12].

The protocol being used to transmit data over the network can also leak information. Wright et al developed a side channel attack to identify the protocol or set of protocols being used in encrypted tunnels. Our focus is to better protect anonymous networks from this types of side channel attack. It has been shown in [13] that this type of attack can break the anonymity of anonymous systems.

2.5 Known Attacks on Tor

After Tor's initial development cycle, researchers have continually been examining the network to see how it resists or deters different forms of traffic analysis. This scrutiny has led Tor into a cycle where threats are identified and removed or hindered in some way. Various researchers developing an attack on the network is a large part of the identification process. The attacks can be divided into two different groups, active or passive. Active attacks involve the attacker manipulating the network in some way, whereas passive attacks do not.

2.5.1 Active Attacks on Tor

One of the first attacks on Tor performed by Murdoch and Danezis used traffic analysis and congestion [29]. The attacker would flood a Tor router with data and watch to see which circuits had corresponding delays in their traffic. This attack assumes the added delay is due to the router processing the extra data from the attacker and not due to other delays in the circuit. By systematically flooding Tor routers, it is possible to determine which routers are used for a circuit. In [20], it is

shown that this type of attack is no longer possible as the number of Tor routers and traffic has grown substantially since the attack was first published.

Tor is also subject to application level attacks. Many websites use java-script, flash, cookies or other content that can request content without going through the Tor network. Tor has taken steps to eliminate these types of threats by creating a TorButton which disables many of these features [4]. Tor also suggests a proxy, such as privoxy or polipo, to direct traffic through Tor [3].

Hackers also pose a threat to Tor. In early January 2010, Tor discovered that two of its directory servers had been compromised [17]. If the hackers had control of a majority of the directory servers, then all of Tor's circuits could be compromised. Tor has taken steps to improve their security since discovery of this attack. Metasploit has also released details of an attack patching a Tor server to discover certain types of traffic and expose those individuals [31].

2.5.2 Passive Attacks on Tor

Zhu et al showed it is possible to correlate the entrance and exit times of Tor using temporal and frequency domain-based techniques [43, 42]. Traffic flowing into a router could be matched to traffic flowing out of another router using only the time a packet arrived or exited. A maximum likelihood approach was used to correlate the two flows. One weakness of this attack is that it requires attackers to observe each entry and exit node in Tor.

Craven developed a method similar to Zhu's but correlates traffic flows leaving one user and arriving at another [13]. This approach uses hidden Markov models and confidence intervals to express the relationship between the two flows. Results from this research shows the attacker can distinguish between different protocols and

different instances of the same protocol. While this approach does not require the attacker to observe each entry and exit node in Tor, it does require the attacker to suspect which two users are communicating beforehand.

Other approaches attempt to break anonymity by exploiting Tor's path selection algorithm. Murdoch and Watson have found it is possible for an attacker to control a large portion of Tor's traffic by controlling a large number of routers, or claiming large bandwidths [30]. By increasing the amount of traffic under attacker control, the attacker increases the probability of being the first node in the circuit. Then the attacker can perform one of the timing attacks mentioned above to determine the destination. Tor has taken steps to counteract this type of attack by allowing users to pick a set of entry guards to be the first node in their circuits [11]. Also, Tor has incorporated a reputation based system into its directory servers to prevent routers from claiming a large bandwidth to attract a larger portion of traffic [5].

2.6 CSSR Algorithm

The network we are proposing will protect Tor from attacks like Craven's [13]. This attack uses a version of CSSR (Causal State Splitting and Reconstruction), to build a HMM (hidden Markov Model) from a discrete series of time stamps. The algorithm was first proposed by Shalizi and Crutchfield [37, 38] and later refined by them [40, 39]. Schwier has further improved the algorithm [34, 35].

2.6.1 Original CSSR Algorithm

The algorithm requires the data series from the model we are trying to reconstruct and the maximum string length to consider, L. The value of L determines how many previous symbols the current symbol is dependent on. The algorithm begins by

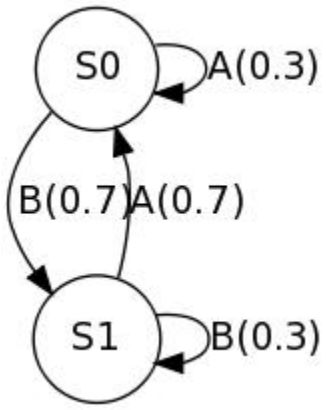

Figure 2.4: Example of a CSSR generated hidden Markov model

splitting the data series into segments of length $L-1$. These represent the different states of the model. It finds the transition probabilities between the states by simply counting how often one state transitions to the next. The transient states can then be removed from the model, as shown in Figure 2.4.

Shalizi makes two important assumptions in the algorithm [37]. First is the user knows the best value for L, which is the maximum subsequence length consider when inferring the model. This assumption means the model returned by the algorithm will be the best possible model for the data sequence. The second assumption is the user has enough data to successfully reconstruct the model, meaning the data contains enough observations of the states and transitions to generate a statistically accurate model.

2.6.2 Removing Assumptions from CSSR

Schwier removes the first assumption about L from the algorithm by iteratively increasing L until the models returned by successive iterations converge [34]. The models converge when states and transitions of the model match, and when the data

produces the same state sequences when traced through each model. By removing this assumption, no prior knowledge of the system and its parameters is needed to generate a HMM modeling the system.

The second assumption is impossible to remove unless we have an infinite amount of data. We will only have finite amounts of data, so we cannot ensure that there are enough observations of each state and transition. Schwier proposes the use of the z-test to calculate the statistical confidence of the model. By using this metric, we can determine the validity of the model or how much more data needs to be collected to successfully model the system.

2.6.3 Improvements for CSSR

The use of confidence intervals to match data to the model in CSSR makes the algorithm better for identification purposes [9]. Shalizi had used maximum likelihood for this purpose in the original algorithm, which is better for classification purposes. This technique has problems with underflow[1], as shown in Figure 2.5. We are using confidence intervals since it is possible to reject all of the transitions and say the data does not fit the model. Confidence intervals also allow us to calculate the false negative rate, at the cost of a slightly higher false positive rate than from maximum likelihood.

Schwier also determined what the best window size for the data would be [34]. This is of special importance to us since we have no prior knowledge of when the users might switch protocols. Schwier found that there is a minimum length of data that is needed to distinguish between two different models, and this minimum length should be used for the window size. If the window were any smaller, we would be unable to

[1]The result of the operation is smaller in magnitude than the smallest value representable by the data type.

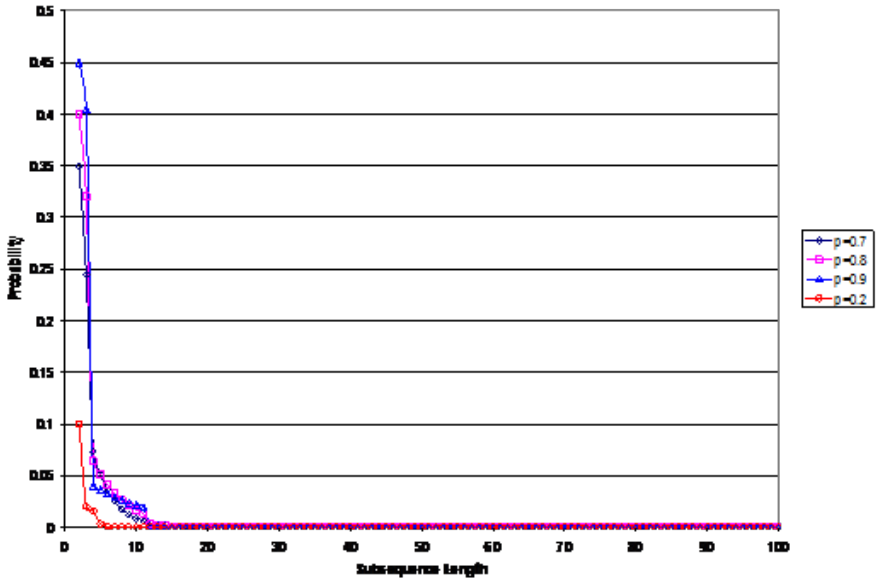

Figure 2.5: Underflow from maximum likelihood

say which model the data matches. If the window were any larger, we would waste data observations and time noticing the switch between the two models. The window sizes needed to distinguish between the two models in Figure 2.6 is given in Table 2.1.

Win Size	Trans $\leq$ 5% overlap	Trans $\leq$ 1% overlap	Type
15	0	0	
16	1	0	Minimum necessary
40	3	1	Minimum sufficient
80	5	3	
160	7	5	
221	8	7	Maximum sufficient
428	8	8	Maximum necessary

Table 2.1: Transitions required to distinguish between models

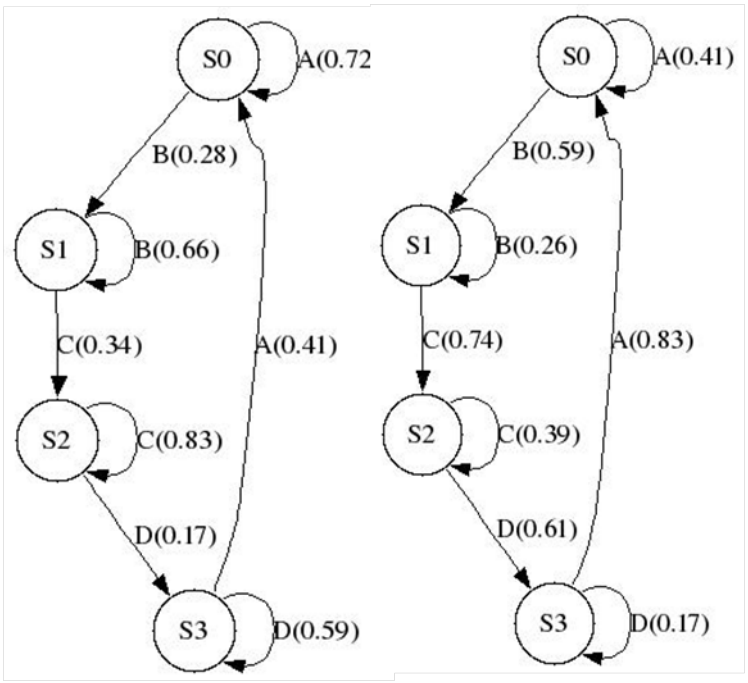

Figure 2.6: Two different models with same state structure

Chapter 3

Theoretical

Anonymity is gained by making it harder to distinguish between different protocols. We constructed a model that increases the variance for different classes of delay and allows individual delays to be disguised. This chapter explains the model and its effect on protocol delays in detail.

3.1 New Anonymity Model

The network we propose is an overlay network using OnionCat for its transport layer, which in turn uses Tor for its transport layer. This overlay network will strengthen the anonymity that Tor provides by eliminating vulnerabilities to some side channel attacks. The network architecture for our network is shown in Figure 3.1.

The network will create multiple paths from the source to the destination through several nodes within the network. These nodes will be used in a way similar to Tor routers in the Tor network. All of these nodes will use OnionCat as a transport layer. These paths are multiplexed into a single channel that the source can use to send a stream of packets to the destination, as shown in Figure 3.2. The network will

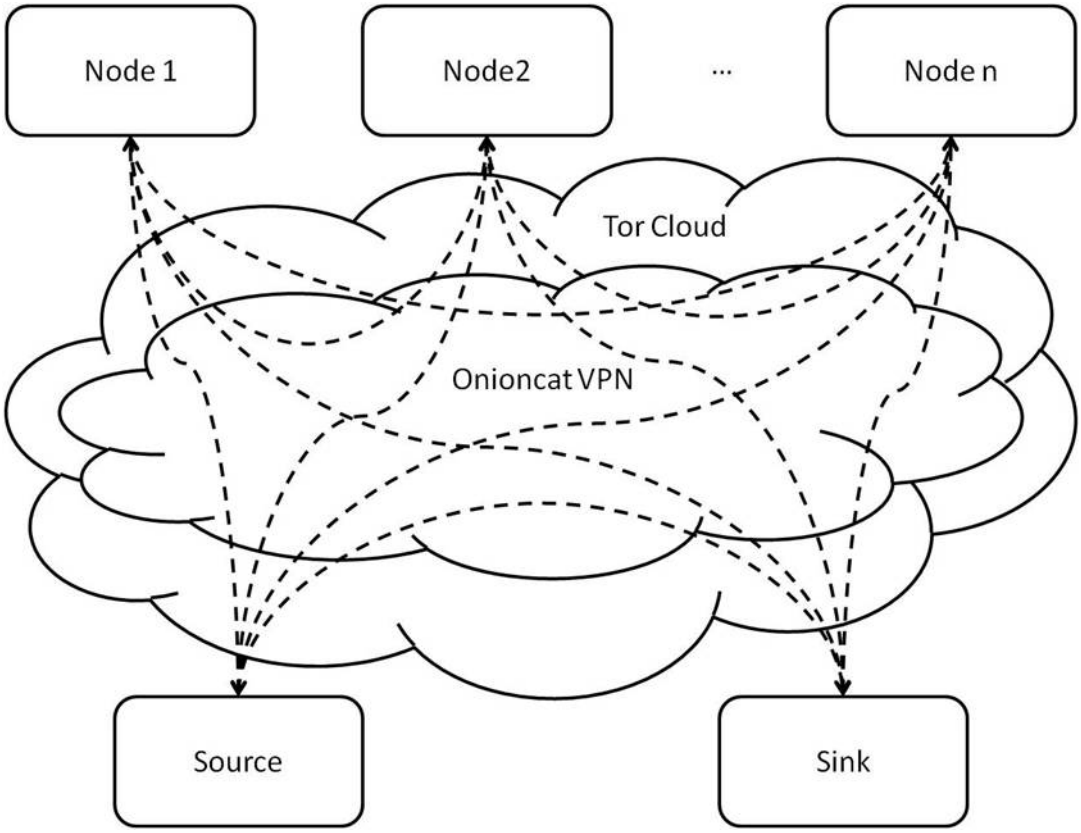

Figure 3.1: Network architecture supporting multiple paths through Tor

choose a random path for each packet it sends over the channel.

By using the OnionCat VPN, it is easy to add or remove nodes from our network. Further, any node could be a possible source or destination. It is possible to create a large number of paths through the network with a small number of nodes, as described in section 3.3. Each of these paths will be characterized by a different delay as shown in section 3.4. The differences between delays will be added to the protocol delay and obscure the relationship between the inter-packet departure and arrival times at the source and destination, as shown in section 3.5.

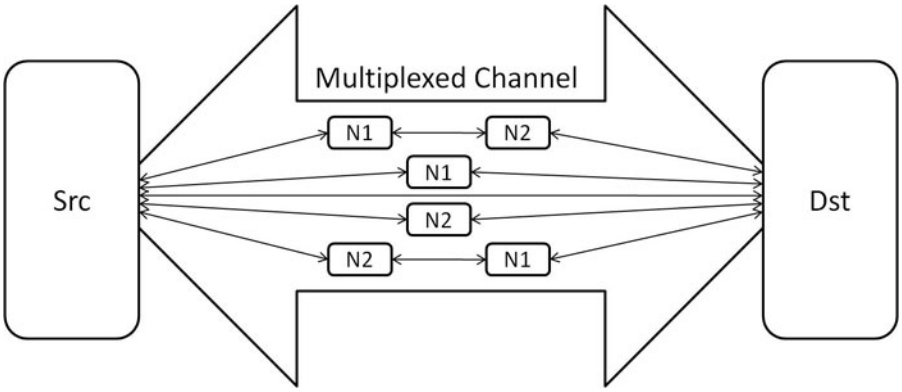

Figure 3.2: Multiple paths multiplexed into a single channel

3.2 Classification Error

In any classification problem, the optimal solution labels all samples correctly. This solution is rarely obtained due to overlap between the various classes of data. The classification error is

$$E = \sum_{\forall i} e_i \times c_i \tag{3.1}$$

where e_i is the number class i samples misclassified and c_i is the cost per sample misclassified of class i. When the cost of misclassification is 1 for each class, the classification error is the number of samples misclassified.

3.2.1 Binary Classifier

Consider a binary classifier which labels samples produced by two Gaussian processes. Since Gaussian random variables are continuous, the two distributions will overlap causing some classification error. Also, the cost of misclassifying each sample is 1 regardless of which Gaussian produces it. To minimize the classification error, we need to minimize the number of misclassified symbols. We can accomplish this by assigning each sample to the Gaussian it is most likely from. This is the Gaussian

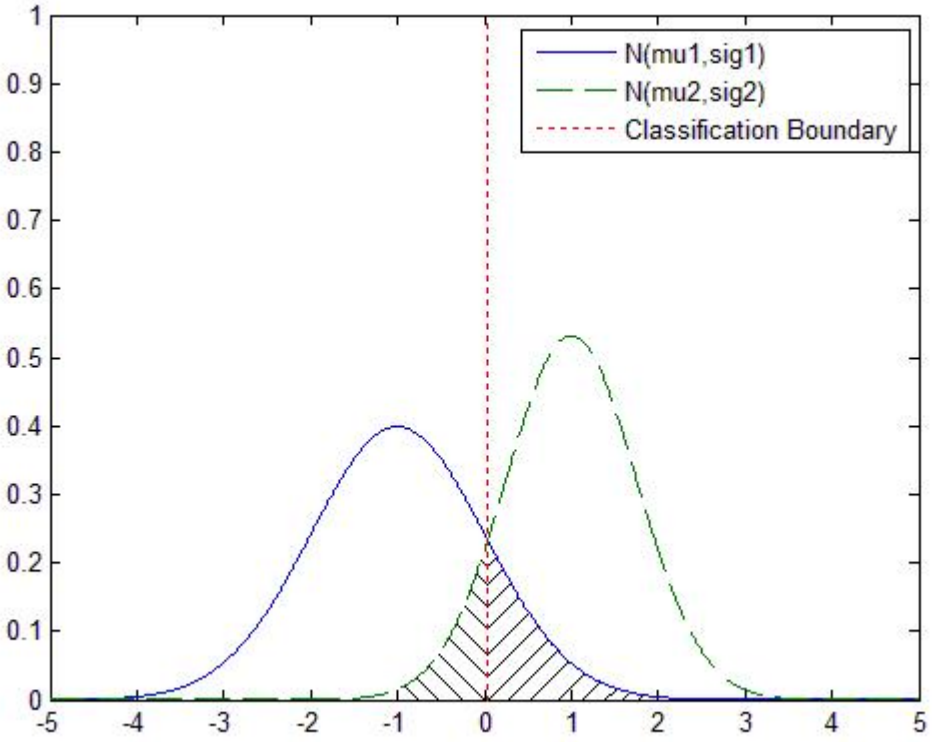

Figure 3.3: Classification of two Gaussian random variables

whose pdf is greater at the sample point.

Figure 3.3 shows a graphical representation of the error as the shaded part under each Gaussian. Due to the mathematical nature of Gaussian random variables, we can find decision boundary for any pair. Once the decision boundary is known, we can find the percentage of samples that will be misclassified.

3.2.2 Binary Classifier Error

We begin finding the error by finding the decision boundary for the pair of Gaussian random variables. The decision boundary is found by finding all points

where the two probability density functions are equal. This point is at

$$x = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} \tag{3.2}$$

when the variance of the two Gaussian random variables are equal and at

$$x = \frac{\pm\sigma_1\sigma_2\sqrt{2(\sigma_1^2 - \sigma_2^2)\ln|\sigma_1/\sigma_2| + (\mu_1 - \mu_2)^2} - (\mu_1\sigma_2^2 - \mu_2\sigma_1^2)}{\sigma_1^2 - \sigma_2^2} \tag{3.3}$$

when the variances differ.

Using the standard score for each distribution we can determine the percentage of samples that will be misclassified for each Gaussian. Of more interest, we can determine the relationship between the mean and variance of the two Gaussian random variables that will have a small amount of error. For a 5% error rate, 2.5% of the samples on each end of the Gaussian can be misclassified. A standard score of ± 1.96 corresponds to a 2.5% error.

This error rate will be achieved when the relationship between the means and variances meets the following criteria

$$|\mu_2 - \mu_1| \geq 3.92\sigma \tag{3.4}$$

when the variance of the two Gaussian random variables are the same or

$$1.96\sigma_1 + \mu_1 \leq \frac{\pm\sigma_1\sigma_2\sqrt{2(\sigma_1^2 - \sigma_2^2)\ln|\sigma_1/\sigma_2| + (\mu_1 - \mu_2)^2} - (\mu_1\sigma_2^2 - \mu_2\sigma_1^2)}{\sigma_1^2 - \sigma_2^2} \tag{3.5}$$

when they are different. A full derivation of the classification error and the relationship between μ and σ is given in the appendix.

3.2.3 Applied to Anonymity

By analyzing different protocols and the delays associated with them, we can determine which anonymous protocols are likely to resist the side channel attack developed in [13] and provide true anonymity. This analysis also aids development of future protocols to determine if they will stand up to these types of side channel attacks.

3.3 Number of Routes to Destination

Our approach also makes side channel attacks more difficult by increasing the number of routes to the destination. Previously, an attacker watching the receiver's incoming traffic would see our communication flow in from a single Tor node. Now, the attacker will see our communication flow in from multiple Tor nodes, and have no reliable way to figure out how all the flows are connected.

Each node has a link to every other node, and packets can travel from node to node until it reaches the receiver. Once a packet has travelled to a node, it cannot travel back to it. It is useful to understand how the number of nodes will effect the number of paths from source to receiver. To begin, there will always be the direct path, which goes directly from the source to the receiver. Then there will be all of the paths that travel from the source through one intermediate node and then to the receiver. And then through two intermediate nodes and so on up until all intermediate nodes are used.

Assume we have n nodes in our system. We can use permutations to calculate the number of paths of length l. A path travelling through one intermediate node will have a length of 1 and so forth. For each length l, the number of paths of that

Nodes	Paths of Length l							Paths
	l=0	l=1	l=2	l=3	l=4	l=5	l=6	
0	1	0	0	0	0	0	0	1
1	1	1	0	0	0	0	0	2
2	1	2	2	0	0	0	0	5
3	1	3	6	6	0	0	0	16
4	1	4	12	24	24	0	0	65
5	1	5	30	60	120	120	0	326
6	1	6	30	120	360	720	720	1957

Table 3.1: Total Number of Paths

length is

$$p_l = \frac{n!}{(n-l)!} \tag{3.6}$$

To calculate the total number of paths from source to receiver, we simply add the number of paths of length 0 up to length n.

$$p = \sum_{l=0}^{n} p_l = \sum_{l=0}^{n} \frac{n!}{(n-l)!} \tag{3.7}$$

The number of paths that result from up to six nodes is given in Table 3.1. The table also shows how the number of paths grows faster than an exponential. This characteristic is nice in that it takes only a handful of nodes to produce a large set of paths.

3.4 Delays of Routes

One of the benefits of having multiple paths from the source to the destination is that we no longer have to rely on a single path to transmit all of our data. When using Tor, all of our traffic will flow across a single path through the network. As such, we can model the delay in the network as a single Gaussian random variable

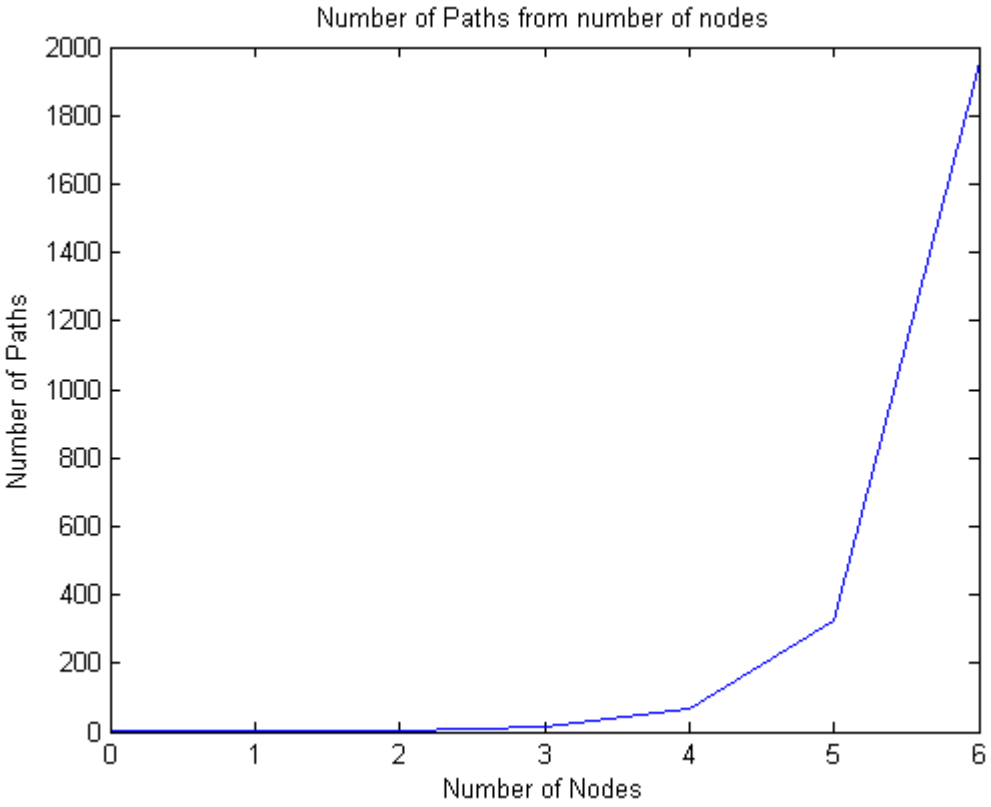

Figure 3.4: Number of paths versus number of nodes

with some delay μ and some variance σ. This model is not the best fit for the delay of the network, but it was chosen because of the simplicity of adding Gaussian random variables together and because the simplification favors the attacker. The attacker will do worse in practice where the heavy-tails cannot be removed.

An attacker can use this delay model to match traffic leaving the sender and arriving at the destination. This type of attack depends only on the variance of the network delay. In [13], it has been shown that the variance of delay through a local Tor network is not large enough to hinder this attack. We have replicated this attack on our local Tor network and on the global Tor network. Furthermore, if this type of attack fails using a Gaussian random variable to model the delay, it will fail if some heavy-tailed model is used.

In our approach, multiple paths are created from the sender to the destination. The delay of the path will strongly depend on the length of the path. Longer paths will have a larger average delay and variance than shorter paths because more Gaussian random variables will be added together. We now have several different paths to deliver our packets to the destination, but still remain vulnerable to the attack in [13].

The novelty of our approach is that it will pick a random path to send each packet to the destination. In a way, we are multiplexing the many paths we have generated into a single channel to the destination. The benefits of this new channel is that its delay will have a different mean and variance for consecutive packets.

3.5 Expected Effects on Inter-Packet Times

The channel will also greatly obscure inter-packet arrival times at the destination. Packets flowing down one path from source to destination have two main factors

that contribute to the inter-packet arrival times. The first is the delay between packets in the protocol, and the second is the variance of the channel. The average delay of the channel has no effect because it is constant for all the packets.

By allowing the delay of the channel to change for each packet, we are adding a third factor to the inter-packet arrival time. Consider a protocol that has two distinct delays between packets, μ_1 and μ_2. We can symbolize these delays as A and B. Now, consider two different connections, c_1 and c_2, characterized by a mean and variance. Let the mean of one channel be μ_3 and the mean of the other channel be $\mu_4 = \mu_3 + (\mu_2 - \mu_1)$. The difference between the channel delays is the same as the difference between the protocol delays.

If all of the packets are sent down c_1, then we can match the inter-packet departure times at the source with the inter-packet arrival times at the destination. If packets choose randomly between c_1 and c_2, then we will not be able to match these attributes at the source and destination. This is because the delays associated with the protocol are not the only delays observed at the destination. The destination will see the delay from the protocol plus one of four possible delays from the different combination of channels. These delays are shown in Figure 3.5.

Two of the additive delays from the channel will have a mean of zero, but will have different variances. These two delays are associated with both packets choosing the same channel. The other two delays result from packets choosing a different channel from the previous packet. These delays have a non-zero average delay. One will be positive and the other negative. Also, one delay is the reflection of the other across the y-axis.

Going back to the example, the inter-packet departure times at the source will consist of the symbols A and B. At the destination, the inter-packet arrival times will consist of the symbols A, B, C and D. Symbol C is symbol A plus the negative non-

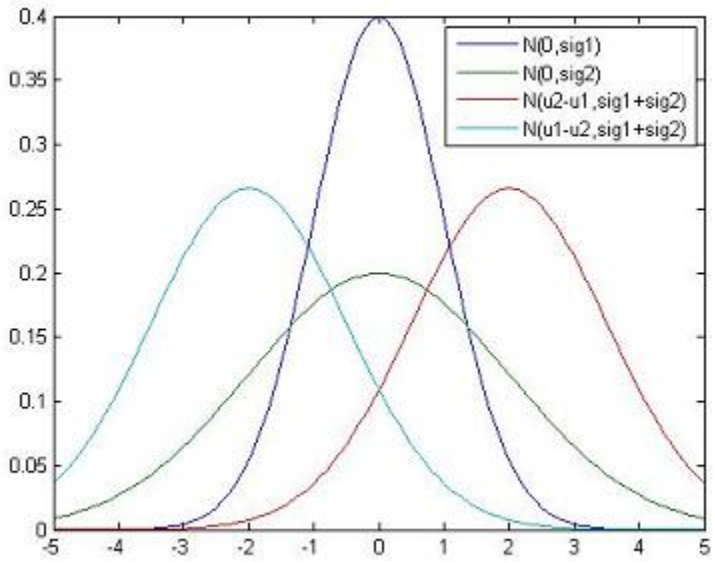

Figure 3.5: Inter-arrival time distributions for 2 paths

zero delay from the channel. Symbol D is symbol B plus the positive non-zero delay from the channel. The addition of the two extra symbols is not enough to confuse a smart attacker though.

What will confuse the attacker is packets symbolized by B at the source having the negative non-zero delay from the channel added and being symbolized by A at the destination. Similarly, packets symbolized by A at the source can be symbolized by B at the destination. There are also many other combinations that could result to confuse the attacker. If we have a large amount of paths from source to destination with different delays, then it is possible to have any inter-packet departure time symbol at the source look like any different inter-packet arrival time at the destination.

3.6 Effects on CSSR Algorithm

In addition to confusing the attacker by allowing certain protocol delays to look like different protocol delays at the destination, our network will also create delays not represented in the protocol. These new delays will need to be symbolized by a new symbol and increase the complexity of the CSSR algorithm. This increase in complexity affects the attacker in three different ways.

The complexity of CSSR is $O(k^{L+1})+O(N)$, where k is the size of the alphabet, L is the maximum subsequence length considered, and N is the size of the input symbol sequence. Given a stream of symbols γ, of fixed length N, from alphabet A, the algorithm is linear in the length of the input data set, but exponential in the size of the alphabet [34].

3.6.1 Increasing the Size of the Alphabet

By allowing packets to travel down different paths to the destination, the inter-packet delays of the protocol have an additive noise from which path the packet chose. Since the delays for the paths are different, additional symbols are required to represent the new observed delays. This increases the base of the exponent, k.

In increasing k, the first term of the complexity equation increases much more rapidly, meaning it will take the attacker more time to generate the hidden Markov model for traffic arriving at the destination. In the simple example from section 3.5, this will double k and the first term will take twice as long to compute.

3.6.2 Increasing the Maximum Subsequence Length

In systems where the inter-packet arrival times are influenced only by the protocol delays, L is usually 1 or 2. This is because the delays of the packets are

independent from one another. In our network, the delay observed for one packet will depend on the delay observed of the previous packet. If packets start arriving out of order, a packet's delay could depend on multiple previous packets.

Since the observed packet delays are no longer independent from one another, the hidden Markov model will not converge until higher subsequence lengths are calculated. This directly increases the exponent L in the first complexity term and greatly increases computational time for the attacker.

3.6.3 Increasing the Number of Input Symbols

A side effect of splitting one symbol from the source into multiple symbols at the destination, is that the attacker will have to collect more data to have the same level of confidence in the results from the CSSR algorithm. This increases the amount of time that the attacker will have to monitor traffic at the destination and also increases the amount of time to process the data with the CSSR algorithm.

Chapter 4

Experimental

This chapter justifies our choice to use Gaussian random variables as models for delays and gives a detailed description of our process implementing the theoretical channel developed in Chapter 3. We begin by collecting samples of delay though OnionCat and comparing them to the standard normal distribution. We finish by showing the resulting models from the CSSR algorithm produced from running our new channel with different number of nodes.

4.1 Delays in OnionCat

We are interested in knowing the average network response time and variance for OnionCat connections. This information will allow us to determine what types of protocols are vulnerable to timing side-channel attacks and which protocols our new channel can protect.

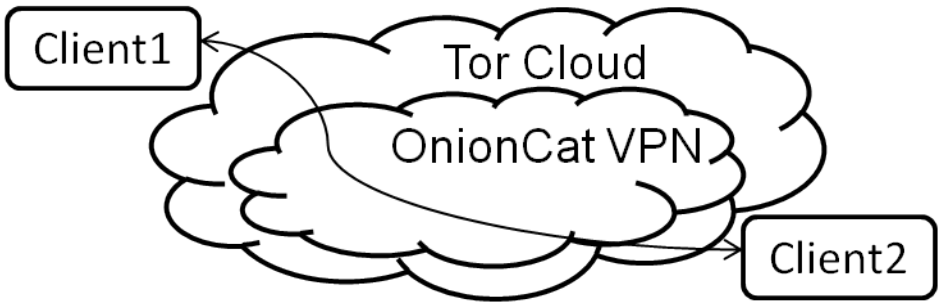

Figure 4.1: Capturing data pings between OnionCat clients

4.1.1 Experimental Setup

The experiment is designed around the network ping tool. Since OnionCat creates a transparent VPN contained within the Tor network, pings can easily be sent through the network. Tor blocks ICMP packets so this approach cannot be used with Tor connections.

To collect the data, we had several clients using OnionCat ping each other using the network interface associated with OnionCat. The process resembles clients as connected in Figure 4.1. The global Tor network was used as the transport layer for OnionCat in these experiments.

4.1.2 Data and Results

Data was collected over three days in the morning, afternoon and evening. The response times were heavy tailed as expected with an average time of 3145 ms, median time of 1916 ms and variance of 4740 ms. The data for the nine collections is given in Table 4.1. The data samples from the first collection are also normalized as compared to the standard normal distribution in Figure 4.2.

This data suggests our proposed channel will be effective since the median

Mean	Variance	Median
3238	4891	1905
3048	4752	1974
2985	4911	1882
3326	4638	1924
3593	4703	2012
2871	4594	1929
3117	4617	1873
3082	4732	1848
3050	4827	1896

Table 4.1: OnionCat Response Times (ms)

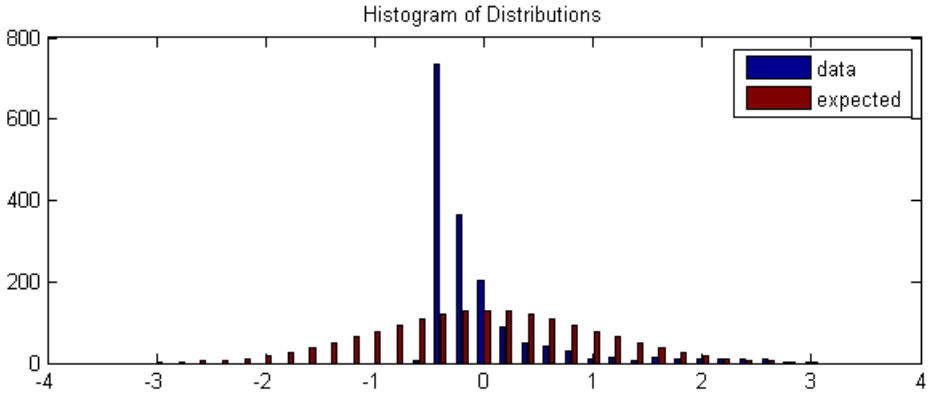

Figure 4.2: Normalized OnionCat Ping Times Compared to Standard Normal Distribution

times differ by up to 164 milliseconds. These differences will obscure the delays of the protocol and destroy the correlation between inter-packet times at the source and destination. We can use our delay model and variance from the collected data to determine the minimum separation needed between protocol delays for timing side channel attacks to be successful. If the protocol delays do not have enough separation, the variance of the network connection through OnionCat will be enough to prevent timing side channel attacks. We will increase the amount of separation further when testing our channel to account for variance in the protocol delay.

Figure 4.2 also suggests our Gaussian model for network delay is acceptable. The collected data fails tests comparing it to the normal distribution, such as the χ^2 or Kolmogorov-Smirnov test. If we ignore the heavy tail of the data, the model fits the data better. We can use our model to calculate the upper bound for the attacker's success rate. In practice, the attacker will do worse than this bound since the attacker cannot ignore the heavy tail found in the data.

4.2 Testing Our Channel

4.2.1 Experimental Setup

The experiment uses the private local Tor network as the network layer for OnionCat. Nodes are connected as shown in Figure 3.1. All of the nodes are running OnionCat to connect to the other nodes. We have designated two nodes to act as the source and the sink and up to three other nodes to serve as different nodes in the paths. All of the possible paths are then constructed for the set of nodes.

The source generates a stream of packets to send to the sink using the protocol shown in Figure 4.3. The delays for each transition are given in Table 4.2. Each packet

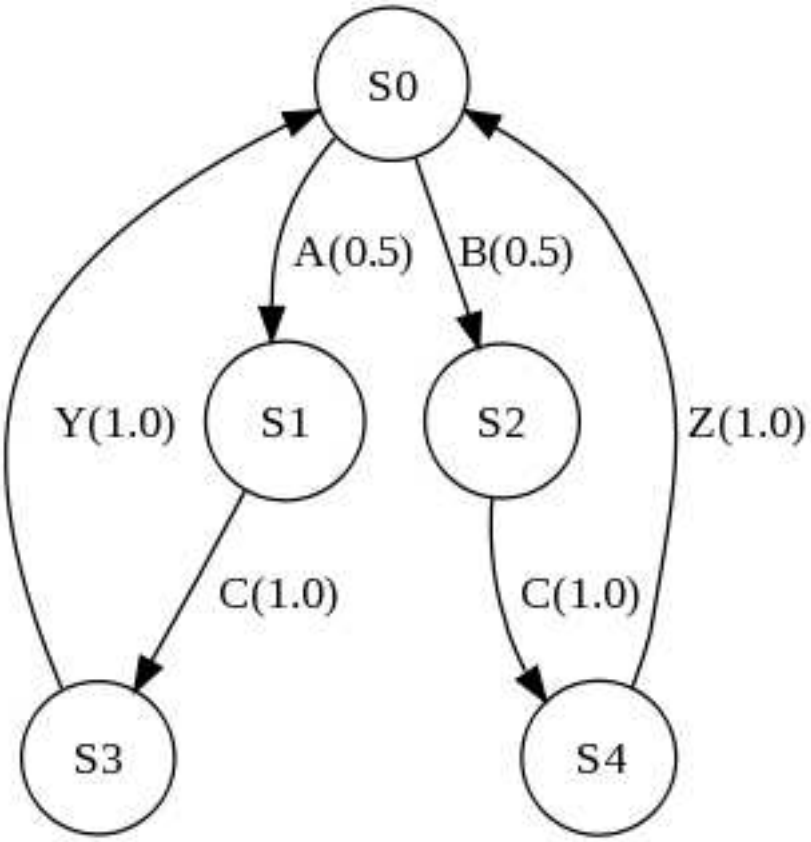

Figure 4.3: 5 state model used for delay

is transmitted on a random path to the sink. The sink's only function is to receive the traffic. No requests or replies are sent back to the source.

We observe two traffic streams, one exiting the source and the other arriving at the sink. The traffic stream at the source depends only on protocol delays and the traffic stream at the sink depends on protocol delays and network delays. The timing side-channel attack is run to attempt to correlate the two streams. If the streams cannot be correlated, our channel has provided protection against the attack. The stream at the sink is also compared to the delay model to determine if the protocol can still be detected.

Symbol	Delay (ms)
A	70
B	140
C	35
Y	105
Z	175

Table 4.2: Delays used by sender (ms)

4.2.2 Data Capture

Data is captured using Wireshark, a network protocol analyzer that allows the user to both capture and browse network traffic at the packet level. Alternatively, tshark may be used. tshark is the command line version of Wireshark.

OnionCat creates a tun or tap network interface for applications to use. We configure Wireshark to capture all traffic on this interface and filter out the packets of length zero. This removes any ACK packets that are present in the stream. We can further filter based on port number of IPv6 address if necessary.

Wireshark will output the timestamps for each packet in seconds from the start of the data capture. We can then convert these timestamps into the inter-packet arrival or departure times necessary for the side channel attack.

4.2.3 Results with Zero Nodes

When no nodes are used to build paths, only the direct path from source to sink exists. The delays at the source correspond to the delays of the protocol. The delays at the sink correspond to the delays of the protocol plus some network delay. Since there is only one path, there is only one random variable characterizing the network delay. The effects of this delay can be seen in Figure 4.4.

Since the sink is not transmitting anything back to the source, inter-packet

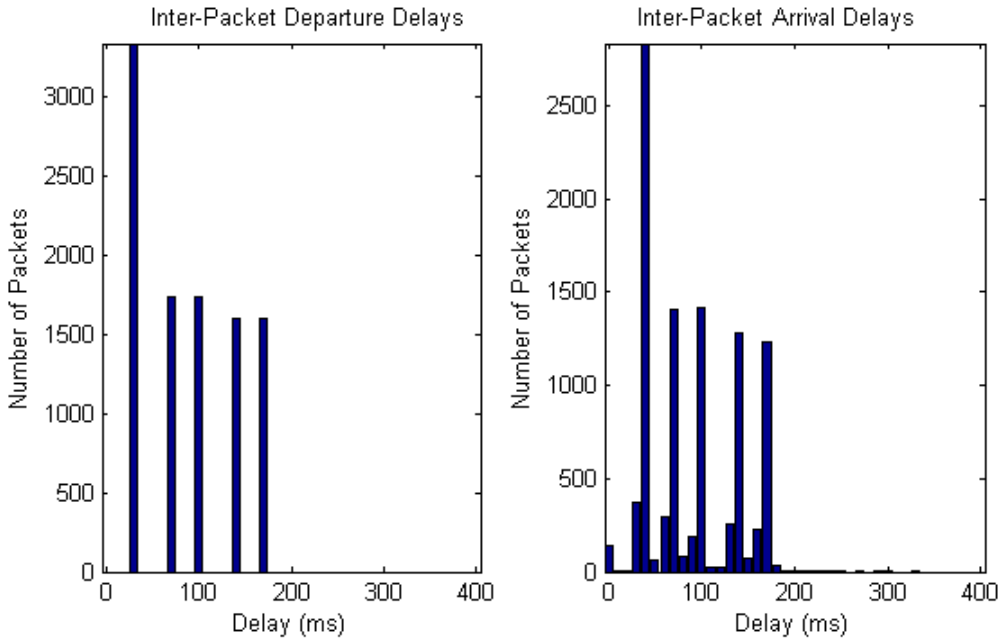

Figure 4.4: Inter-packet delays with 0 nodes

arrival times at the sink can be symbolized as shown in Table 4.2. The model reconstructed from the inter-packet delays leaving the source looks very much like the model shown in Figure 4.3. This symbolization and source model is used for each test case presented below. The model reconstructed from the inter-packet delays arriving at the sink is shown in Figure 4.5. There are some low probability transitions in this model due to the variance of the channel or Tor reconstructing its circuits. These transitions may be pruned out of the model since they correspond to such a low percentage of the traffic.

The acceptance rate is the percentage of windows that match at the source and the sink and have a path through the model. Since the model is derived directly from the protocol being used to transmit the data at the source, 100% of the windows observed at the source will have a path through the model. This means our channel will not hide the protocol the source is using to transmit data. It also means the

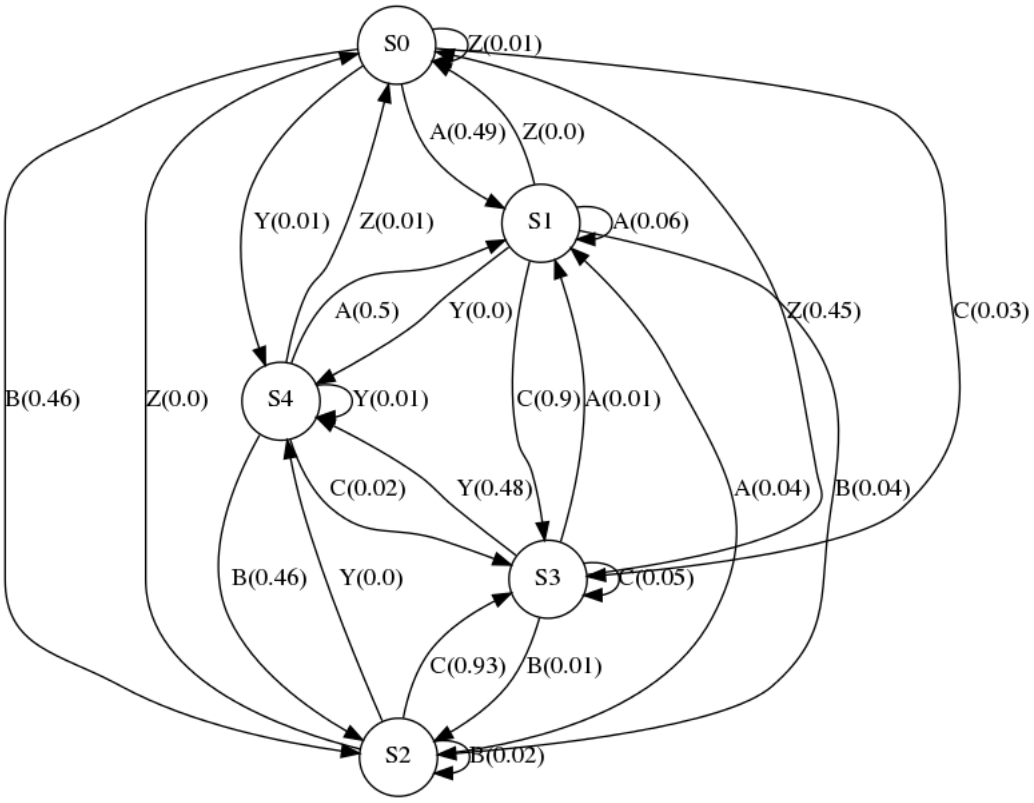

Figure 4.5: Reconstructed model from received delays with 0 nodes

Rate	$w = 2$	$w = 4$	$w = 8$	$w = 16$
Acceptance	0.9320	0.8947	0.8276	0.7046
Rejection	0.0680	0.1053	0.1724	0.2954

Table 4.3: Acceptance and rejection rates with 0 nodes

acceptance rate only depends on how many windows at the sink don't have a path through the model. The rejection rate is the percentage of windows that do not match or do not have a path through the model.

The acceptance and rejection rates with 0 nodes is shown in Table 4.3 for different window sizes. Our channel with 0 nodes is the direct path from source to sink through OnionCat. As expected, it is vulnerable to timing side-channel attacks.

4.2.4 Results with One Intermediate Node

When one intermediate node is used to build paths, two paths from source to sink exist. The delays at the source correspond to the delays of the protocol. The delays at the sink correspond to the delays of the protocol plus some network delay. In this case, there are two random variables characterizing the network delay. There are four different ways these delays can be added to consecutive packets to produce different observed delays at the sink. The effects of this delay can be seen in Figure 4.6.

The model reconstructed from the inter-packet delays arriving at the sink is shown in Figure 4.7. The low probability transitions in this model are slightly higher than the corresponding transitions in Figure 4.5. This suggests that the differences in delays from different paths through the network affect the delays present in the protocol. These transitions account for less than 5% of the total traffic and may be pruned out of the model.

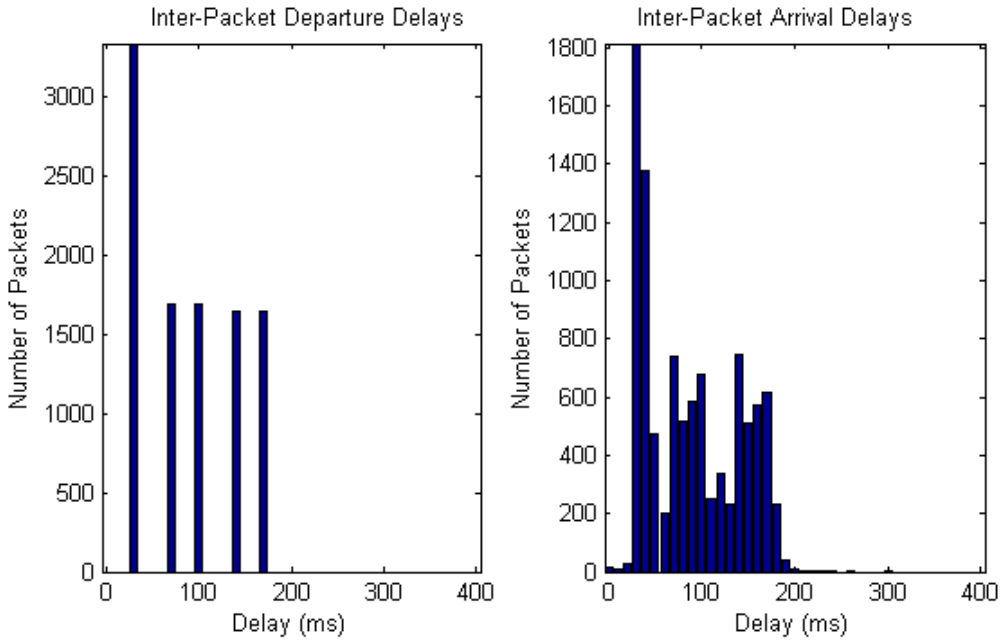

Figure 4.6: Inter-packet delays with 1 node

Rate	$w = 2$	$w = 4$	$w = 8$	$w = 16$
Acceptance	0.9015	0.8245	0.6955	0.4968
Rejection	0.0985	0.1755	0.3045	0.5032

Table 4.4: Acceptance and rejection rates with 1 node

The acceptance and rejection rates with 1 node is shown in Table 4.4. Our channel with 1 intermediate node is still vulnerable to timing side-channel attacks. The acceptance rate does fall slightly suggesting more nodes are needed to provide better anonymity.

4.2.5 Results with Two Intermediate Nodes

When two intermediate nodes are used to build paths, five paths from source to sink exist. The delays at the source correspond to the delays of the protocol. The delays at the sink correspond to the delays of the protocol plus some network delay. In

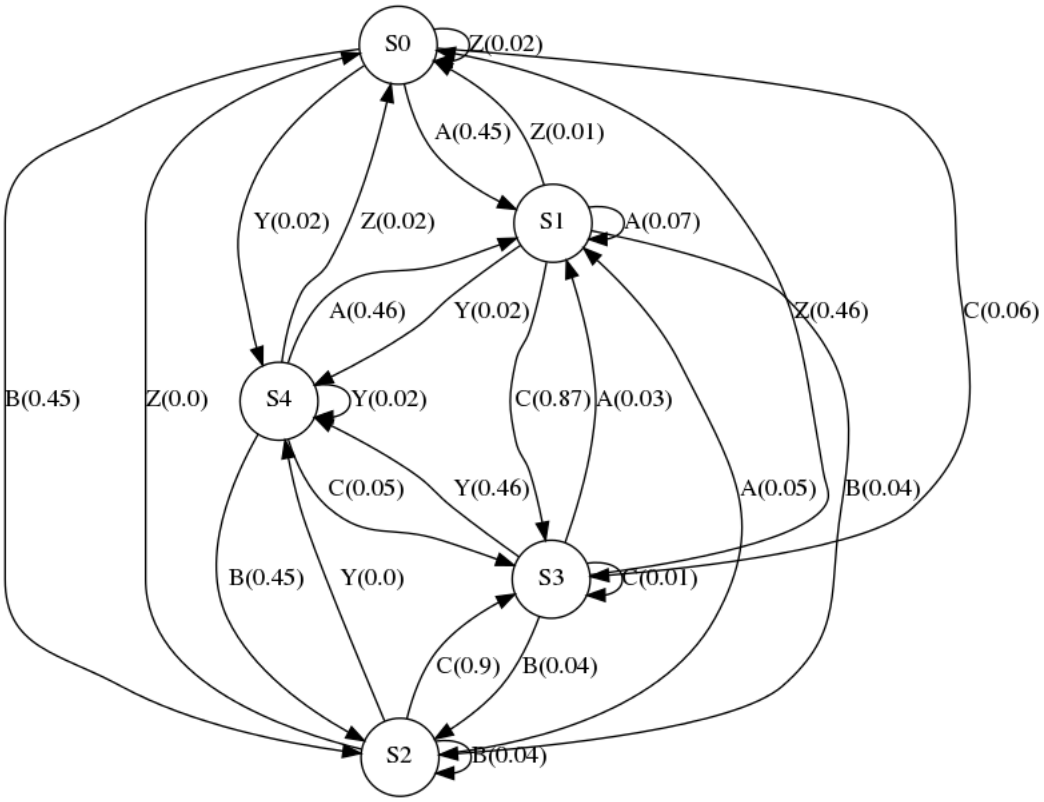

Figure 4.7: Reconstructed model from received delays with 1 node

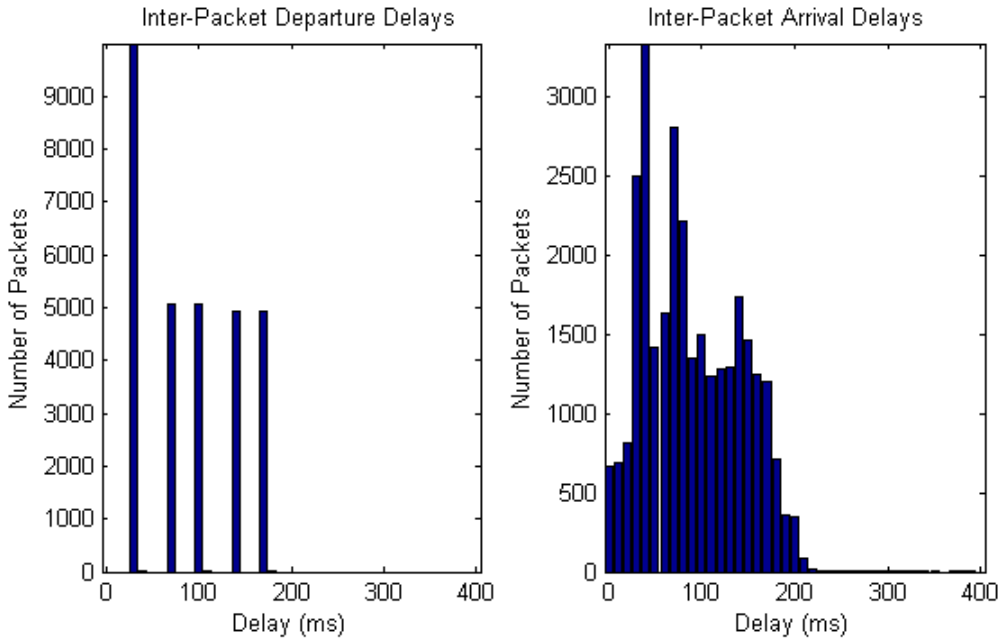

Figure 4.8: Inter-packet delays with 2 nodes

this case, there are five random variable characterizing the network delay. There are 25 different ways these delays can be added to consecutive packets to produce different observed delays at the sink. The effects of this delay can be seen in Figure 4.8.

The model reconstructed from the inter-packet delays arriving at the sink is shown in Figure 4.9. Some of the low probability transitions in this model are slightly higher than the corresponding transitions in Figure 4.7. These transitions now account for a more significant amount of the traffic and cannot be pruned from the model. They show how the delays added from packets taking different paths obscure some of the protocol delays.

The acceptance and rejection rates with 2 nodes is shown in Table 4.5. Our channel with 2 intermediate nodes does a good job of protecting users from timing side-channel attacks. The acceptance rate is just over 50% for the smallest window length. The difference between delays for paths of length 0 and paths of length 2 is

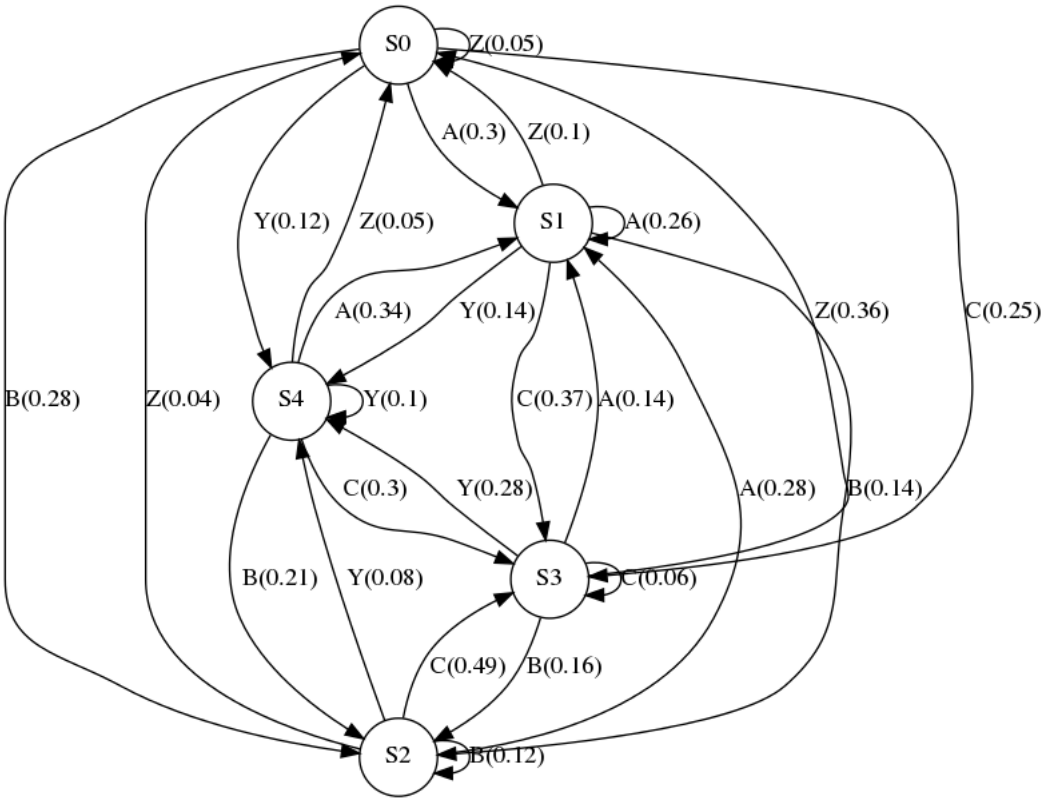

Figure 4.9: Reconstructed model from received delays with 2 nodes

Rate	$w = 2$	$w = 4$	$w = 8$	$w = 16$
Acceptance	0.5247	0.2606	0.0809	0.0084
Rejection	0.4753	0.7394	0.9191	0.9916

Table 4.5: Acceptance and rejection rates with 2 nodes

large enough to make one delay look like another. At this point, packets also started arriving out of order at the sink.

4.2.6 Results with Three Intermediate Nodes

When three intermediate nodes are used to build paths, 16 paths from source to sink exist. The delays at the source correspond to the delays of the protocol. The delays at the sink correspond to the delays of the protocol plus some network delay. In this case, there are 16 random variable characterizing the network delay. There are 256 different ways these delays can be added to consecutive packets to produce different observed delays at the sink. The effects of this delay can be seen in Figure 4.10.

The model reconstructed from the inter-packet delays arriving at the sink is shown in Figure 4.11. Most of the low probability transitions in Figure 4.5 are significantly higher in this model. These transitions now account for an even more significant amount of the traffic and cannot be pruned from the model. They show how the delays added from packets taking different paths obscure many of the protocol delays.

The acceptance and rejection rates with 3 intermediate nodes is shown in Table 4.6. Our channel with 3 intermediate nodes does a better job of protecting users from timing side-channel attacks. The acceptance rate is under 50% for the smallest window size. The difference between delays is large enough to make delays for one

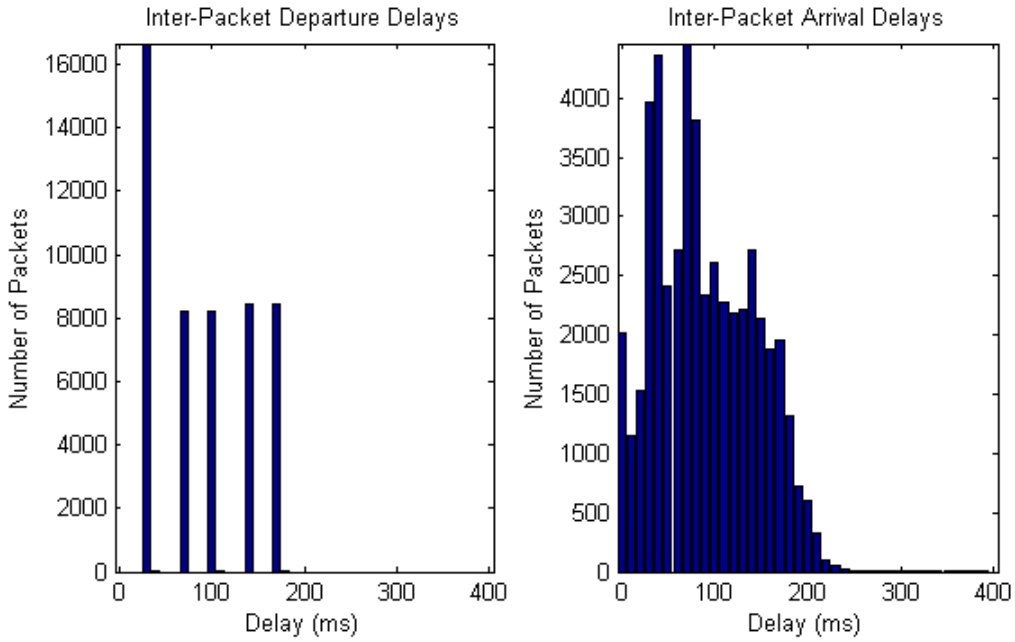

Figure 4.10: Inter-packet delays with 3 nodes

Rate	$w = 2$	$w = 4$	$w = 8$	$w = 16$
Acceptance	0.4284	0.1515	0.0260	0.0013
Rejection	0.5716	0.9485	0.9740	0.9987

Table 4.6: Acceptance and rejection rates with 3 nodes

symbol look like another and make many packets arrive out of order. While reordering the packets at the sink is not a challenging problem, it does present difficulties for the attacker seeing two delays in the protocol represented as a single delay at the sink.

4.2.7 Overview of Results

It is clear from the histograms in Figures 4.4, 4.6, 4.8, and 4.10 that a small number of nodes removes the crisp distinctions between protocol delays observed at the source. This can also be seen in the smallest transition probabilities in Figure 4.5 increasing as more intermediate nodes are added.

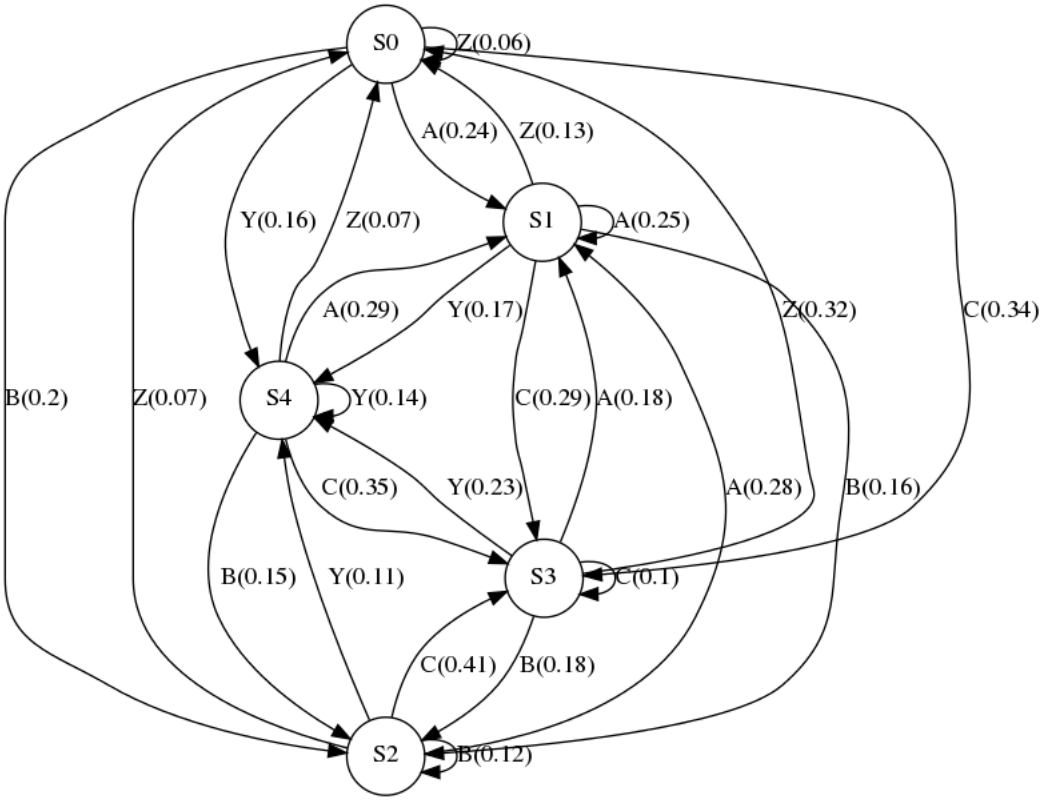

Figure 4.11: Reconstructed model from received delays with 3 nodes

Nodes	$w = 2$	$w = 4$	$w = 8$	$w = 16$
$n = 0$	0.9320	0.8947	0.8276	0.7046
$n = 1$	0.9015	0.8245	0.6955	0.4968
$n = 2$	0.5247	0.2606	0.0809	0.0084
$n = 3$	0.4284	0.1515	0.0260	0.0013

Table 4.7: Acceptance rates for up to 3 nodes

Nodes	$w = 2$	$w = 4$	$w = 8$	$w = 16$
$n = 0$	0.0680	0.1053	0.1724	0.2954
$n = 1$	0.0985	0.1755	0.3045	0.5032
$n = 2$	0.4753	0.7394	0.9191	0.9916
$n = 3$	0.5716	0.9485	0.9740	0.9987

Table 4.8: Rejection rates for up to 3 nodes

The falling acceptance rates and rising rejection rates also support our channel providing anonymity to the users. The acceptance rate is the percentage of window pairs that match and have a path through the model. This represents the attacker's ability to correlate packets leaving the source and arriving at the sink. There are two possible reasons for a pair of windows to be rejected. The first is that the model cannot handle the symbol string in the window, and the second is that the windows do not match each other.

The acceptance rate falls as the window increases because the same corrupted symbol is present in more windows. Over 90% of the windows are accepted with a window length of 2 and 1 or 0 nodes. When 2 or more intermediate nodes are used, then the acceptance rate falls drastically to around 50% or less.

As the acceptance rate falls, the rejection rate increases. The rejection rate is the percentage of window pairs that are different or do not have a path through the model. This represents the percentage of windows that the attacker cannot correlate between the source and sink.

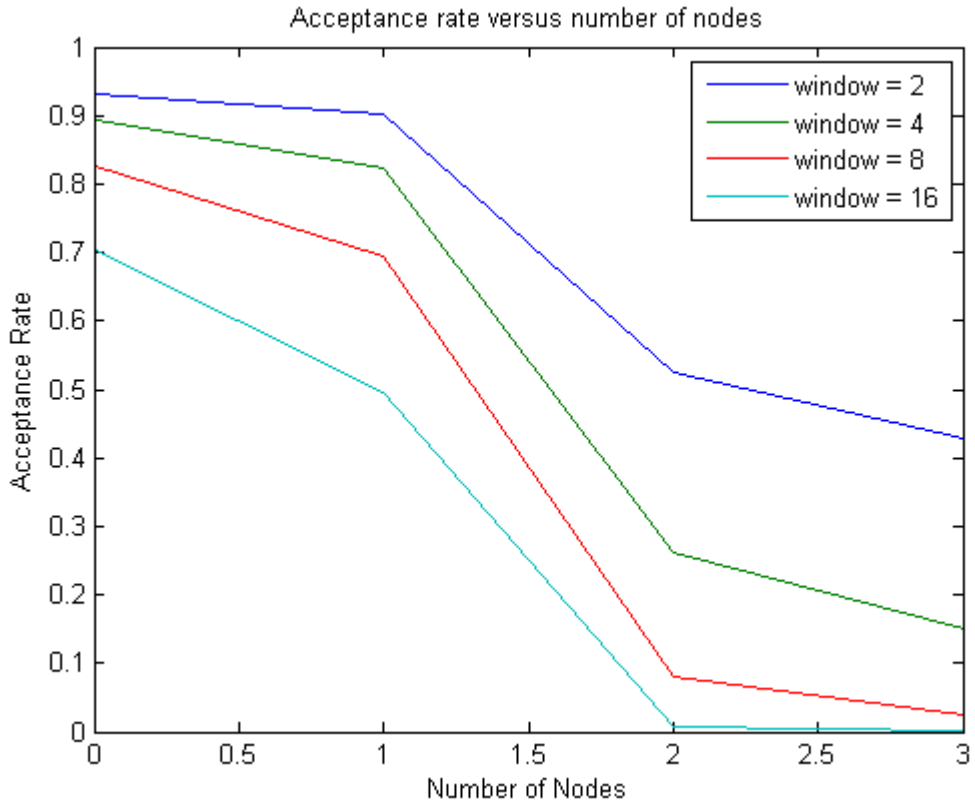

Figure 4.12: Acceptance rates versus number of nodes

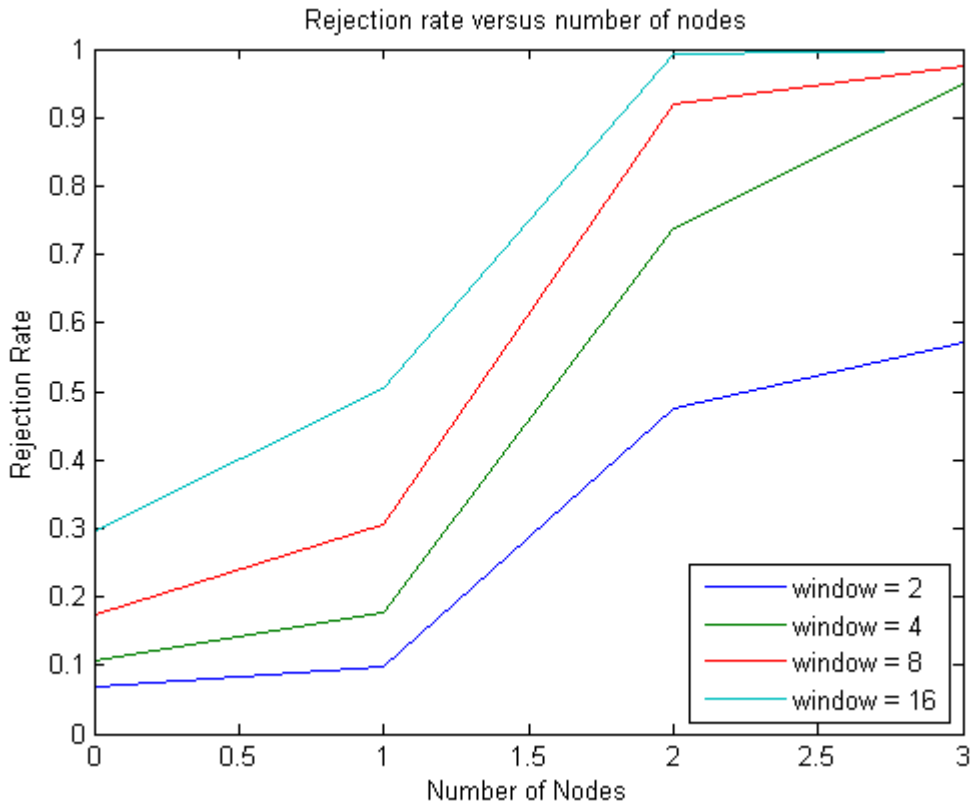

Figure 4.13: Rejection rates versus number of nodes

4.3 Implementation Concerns

The attacker must not be able to determine which path a packet is travelling on, or distinguish between packets travelling to the same node but on different paths when it leaves the source or arrives at the sink. To protect against this, a small number of entry guards should be used to ensure each path does not have a different first hop. Since Tor encrypts the remainder of the path, this should prevent the attacker from determining the path of a packet. This works at both the source and sink because OnionCat uses Tor's hidden services, and both source and sink build a Tor circuit to the rendezvous point.

If the attacker is able to separate one path from all of the other paths, the attacker will be able to perform the timing side channel attack on this path. Each individual path is vulnerable to this type of attack because the network delay is similar for each packet. Also, the delays between packets will be increased since the path does not transmit every packet from the protocol. When three intermediate nodes are used to form paths, this link will carry around 6.25% of the total traffic, but this amount is enough to compromise the user's anonymity. The communication protocol would remain a secret since the attacker can only reliably capture a small percentage of it.

Chapter 5

Conclusions

5.1 Summary

In this work, we examined why low-latency anonymous networks are vulnerable to timing side-channel attacks and investigate ways to eliminate these vulnerabilities. Custom client, server, and forwarding programs using OnionCat and a private Tor network provide the experimental foundation for our work. We showed how a collection of paths can protect this network from timing side-channel attacks.

We determined we could use Gaussian random variables to model the delays in the network. This model ignores the heavy tail, which attackers cannot ignore when physically performing the attack. The results from our mathematical models represent the best case scenario for the attacker. In practice, the attacker will perform worse.

The inter-packet delay signatures at the source and sink were physically captured from our implementation. We collected data with various number of nodes helping to form the paths from source to sink. This data was used by the timing side-channel attack. The results showed how increasing the number of nodes made it

more difficult to correlate the signatures at the source and sink. The effects of adding each node to our channel was also determined.

We found that this type of channel can help strengthen anonymity in low-latency anonymous networks from both local and global adversaries. The channel requires a small number of nodes to produce this anonymity and does not increase the latency considerably. This channel could be integrated into the anonymous network or implemented as an overlay network.

5.2 Recommendations for Further Research

In the process of creating this channel, many interesting ideas and questions came to mind that could not be investigated due to time constraints. These ideas and questions are mentioned here and quickly discussed.

Perhaps the easiest question to answer is how much additional latency would this channel contribute to the anonymous network per node used? Similarly, how many nodes can be used before the extra latency makes the network tedious to use? Various studies have examined how long users are willing to wait for interactive content. We could use these studies to better understand how our channel will impact the network's usability.

Studying the channel's effects on other protocols is another research area. The protocols examined in this work were all artificially created. We could determine what types of protocols and applications would be vulnerable, and which ones our channel can help protect. Any application using TCP could be studied with Tor, but OnionCat would be necessary to study IP based applications.

We can also look at our channel from the attacker's perspective, and find what types of attacks our channel is vulnerable to. We know it helps protect against timing

side-channel attacks, but there are many other possible types of attacks. Could one of these attacks remove the anonymity provide by our channel? This different viewpoint can help us determine new vulnerable areas in anonymous communication networks and mark the beginning of a new attack and defend cycle.

Appendices

Appendix A Derivation of Classification Error

Consider a binary classifier that classifies elements as belonging to one of two normal distributions. The classification error can be minimized by choosing the intersection of the two distributions as the classification boundary. In this appendix, a mathematical derivation is given to find the classification boundary and determine what the error rate is.

A.1 Deriving the Classification Boundary

The pdf for the normal distribution is of the form

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{1}$$

where μ is the mean and σ is the variance of the random variable. The intersections can be found by solving for points where one distribution equals the other.

$$\frac{1}{\sqrt{2\pi\sigma_1^2}} e^{\frac{-(x-\mu_1)^2}{2\sigma_1^2}} = \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{\frac{-(x-\mu_2)^2}{2\sigma_2^2}} \tag{2}$$

$$\frac{e^{\frac{-(x-\mu_1)^2}{2\sigma_1^2}}}{e^{\frac{-(x-\mu_2)^2}{2\sigma_2^2}}} = \frac{\sqrt{2\pi\sigma_1^2}}{\sqrt{2\pi\sigma_2^2}}$$

$$e^{\frac{(x-\mu_2)^2}{2\sigma_2} - \frac{(x-\mu_1)^2}{2\sigma_1}} = \left|\frac{\sigma_1}{\sigma_2}\right|$$

$$\frac{\sigma_1^2(x-\mu_2)^2 - \sigma_2^2(x-\mu_1)^2}{2\sigma_1^2\sigma_2^2} = \log\frac{\sigma_1}{\sigma_2}$$

$$(\sigma_1^2 - \sigma_2^2)x^2 - (2\sigma_1^2\mu_2 - 2\sigma_2^2\mu_1)x + (\sigma_1^2\mu_2^2 - \sigma_2^2\mu_1^2) = 2\sigma_1^2\sigma_2^2 \log(\sigma_1/\sigma_2)$$

$$x^2 - \frac{2\sigma_1^2\mu_2 - 2\sigma_2^2\mu_1}{\sigma_1^2 - \sigma_2^2}x = \frac{2\sigma_1^2\sigma_2^2 \log(\sigma_1/\sigma_2) + \sigma_2^2\mu_1^2 - \sigma_1^2\mu_2^2}{\sigma_1^2 - \sigma_2^2}$$

$$\left(x - \frac{\sigma_1^2\mu_2 - \sigma_2^2\mu_1}{\sigma_1^2 - \sigma_2^2}\right)^2 = \frac{2\sigma_1^2\sigma_2^2 \log(\sigma_1/\sigma_2) + \sigma_2^2\mu_1^2 - \sigma_1^2\mu_2^2}{\sigma_1^2 - \sigma_2^2} + \left(\frac{\sigma_1^2\mu_2 - \sigma_2^2\mu_1}{\sigma_1^2 - \sigma_2^2}\right)^2$$

$$x = \sqrt{\frac{2\sigma_1^2\sigma_2^2 \log(\sigma_1/\sigma_2) + \sigma_2^2\mu_1^2 - \sigma_1^2\mu_2^2}{\sigma_1^2 - \sigma_2^2} + \left(\frac{\sigma_1^2\mu_2 - \sigma_2^2\mu_1}{\sigma_1^2 - \sigma_2^2}\right)^2} + \frac{\sigma_1^2\mu_2 - \sigma_2^2\mu_1}{\sigma_1^2 - \sigma_2^2}$$

$$x = \sqrt{\frac{\sigma_1^2\sigma_2^2\left(2(\sigma_1^2 - \sigma_2^2)\log(\sigma_1/\sigma_2) + (\mu_1 - \mu_2)^2\right)}{(\sigma_1^2 - \sigma_2^2)^2}} + \frac{\sigma_1^2\mu_2 - \sigma_2^2\mu_1}{\sigma_1^2 - \sigma_2^2}$$

$$x = \frac{\pm\sigma_1\sigma_2\sqrt{2\left(\sigma_1^2 - \sigma_2^2\right)\ln|\sigma_1/\sigma_2| + (\mu_1 - \mu_2)^2} - (\mu_1\sigma_2^2 - \mu_2\sigma_1^2)}{\sigma_1^2 - \sigma_2^2} \tag{3}$$

or

$$x = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} \tag{4}$$

when the variance for the two distributions are equal.

All observations less than the classification boundary will be classified as one distribution, and all observations greater than it will be classified as the other. When following this rule, observations will be assigned to the class they most likely belong to. This approach can be used to find the classification boundaries between multiple normal distributions as well.

A.2 Deriving the Classification Error per Class

Once the classification boundary is known for the distributions, the expected number of misclassified elements can be calculated. The standard score is helpful.

$$z = \frac{x - \mu}{\sigma} \tag{5}$$

By converting the classification boundary to one on the standard normal distribution, the number of misclassified elements can be looked up or calculated. The calculation is

$$P(Z \leq z) = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{\frac{-u^2}{2}} du \tag{6}$$

when z is negative or

$$P(Z \geq z) = \int_{z}^{\infty} \frac{1}{\sqrt{2\pi}} e^{\frac{-u^2}{2}} du$$

when z is positive. For a 5% error rate, 2.5% of each class can be misclassified. This assumes the left and right tails of each class will overlap with some other class and are misclassified. The standard score that corresponds to this error rate is ± 1.96

By enforcing the criteria of

$$|z| \geq 1.96 \tag{7}$$

the relationship between the mean and variance of the two distributions can be determined. The 5% error rate will hold when

$$|\mu_2 - \mu_1| \geq 3.92\sigma \tag{8}$$

when the variances are equal or

$$1.96\sigma_1 + \mu_1 \leq \frac{\pm\sigma_1\sigma_2\sqrt{2\left(\sigma_1^2 - \sigma_2^2\right)\ln|\sigma_1/\sigma_2| + \left(\mu_1 - \mu_2\right)^2} - \left(\mu_1\sigma_2^2 - \mu_2\sigma_1^2\right)}{\sigma_1^2 - \sigma_2^2} \tag{9}$$

and

$$1.96\sigma_2 + \mu_2 \leq \frac{\pm\sigma_1\sigma_2\sqrt{2\left(\sigma_1^2 - \sigma_2^2\right)\ln|\sigma_1/\sigma_2| + \left(\mu_1 - \mu_2\right)^2} - \left(\mu_1\sigma_2^2 - \mu_2\sigma_1^2\right)}{\sigma_1^2 - \sigma_2^2}$$

when the variances are different.

Bibliography

[1] Merriam-webster dictionary. http://www.merriam-webster.com/. [Online; accessed February 2011].

[2] Tor project: Anonymity online. https://www.torproject.org/.

[3] The tor project faq. why do we need polipo or privoxy with tor? https://trac.torproject.org/projects/tor/wiki/TheOnionRouter/TorFAQ. [Online; accessed January 2011].

[4] Torbutton. https://www.torproject.org/torbutton/index.html.en. [Online; accessed January 2011].

[5] K. Bauer, D. McCoy, D. Grunwald, T. Kohno, and D. Sicker. Low-resource routing attacks against anonymous systems. Technical Report CU-CS-1025-07, University of Colorado, Boulder, CO. http://www.cs.colorado.edu/department/publications/reports/docs/CU-CS-1025-07.pdf.

[6] O. Berthold, H. Federrath, and S. Köpsell. Webmixes: A system for anonymous and unobservable internet access. In H. Federrath, editor, *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, pages 115–129. LNCS 2009, Springer-Verlag, July 2000.

[7] H. Bhanu. Timing side channel attacks on ssh. Master's thesis, Clemson University, May 2010.

[8] D. Boneh and D. Brumley. Remote timing attacks are practical. In *Proceedings of 12th Usenix Security Symposium*, 2003.

[9] R. R. Brooks, J. M. Schwier, and C. Griffin. Behavior detection using confidence intervals of hidden markov models. *IEEE Transactions on Systems, Man, and Cybernetics*, 39(6):1484–1492, December 2009.

[10] D. Chaum. The dining cryptographers problem: Unconditional senderand recipient untraceability. *Journal of Cryptography*, 1, 1988.

[11] S. Chen, R. Wang, X.F. Wang, and K. Zhang. Side-channel leaks in web applications: a reality today, a challenge tomorrow. In *Proceedings of IEEE Symposium on Security and Privacy*, Oakland, CA, May 2010. IEEE Computer Society.

[12] S. Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. Side-channel leaks in web applications: a reality today, a challenge tomorrow. In *Proceedings of the IEEE Symposium on Security and Privacy (Oakland)*. IEEE Computer Society, May 2010.

[13] R. Craven. Traffic analysis of anonymity systems. Master's thesis, Clemson University, May 2010.

[14] G. Danezis and R. Clayton. *Digital Privacy: Theory, Technologies, and Practices*. Auerbach Publications, Boca Raton, FL, 2008.

[15] G. Danezis, R. Dingledine, and N. Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 2–15. IEEE CS, May 2003.

[16] R. Deibert, J. Palfrey, R. Rohozinski and J. Zittrain, editor. *Access Denied: The Practice and Policy of Global Internet Filtering (Information Revolution and Global Politics)*. The MIT Press, February 2008.

[17] R. Dingledine. Tor project infrastructure updates in response to security breach. `http://archives.seul.org/or/talk/Jan-2010/msg00161.html`. [Online; accessed April 2011].

[18] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, San Diego, CA, August 2004.

[19] M. Dusi, M. Crotti, and F. Gringoli. Tunnel hunter: Detecting application-layer tunnels with statistical fingerprinting. *Communications Networks*, 53(1):81–97, January 2009.

[20] N. Evans, R. Dingledine, and C. Grothoff. A practical congestion attack on tor using long paths. In *Proceedings of 18th USENIX Security Symposium*, pages 33–50, Montreal, Canada, August 2009.

[21] B. Fisher. Onioncat - an anonymous internet overlay. 2009.

[22] A. Hintz. Fingerprinting websites using traffic analysis. In R. Dingledine and P. Syverson, editors, *Proceedings of Privacy Enhancing Technologies workshop (PET 2002)*, San Francisco, CA, 2002.

[23] jrandom. I2p: A scalable framework for anonymous communication. `http://www.i2p2.de/techintro.html`. [Online; Accessed December 2010].

[24] P. Kocher, J. Jaffe and B. Jun. Differential power analysis. *Lecture Notes In Computer Science*, 1666, 1999.

[25] A. Pfitzmann and M. Kohntopp. Anonymity, unobservability, and pseudonymity a proposal for terminology. In Hannes Federrath, editor, *Proceedings of International workshop on Designing privacy enhancing technologies: design issues in anonymity and unobservability*. Springer-Verlag New York, Inc., 2001.

[26] R. Lewin. A signal-intelligence war. *Journal of Contemporary History*, 16(3):501–512, July 1981.

[27] N. Mathewson and R. Dingledine. Practical traffic analysis: Extending and resisting statistical disclosure. In D. Martin and A. Serjantov, editors, *Proceedings of Privacy Enhancing Technologies workshop (PET 2004)*, volume 3424, pages 17–34, May 2004.

[28] U. Möller, L. Cottrell, P. Palfrader, and L. Sassman. Mixmaster protocol – version 2. *IETF Internet Draft*, July 2003. `http://www.abditum.com/mixmaster-spec.txt`.

[29] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of tor. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 183–195, Oakland, CA, May 2005.

[30] S. J. Murdoch and R. N. M. Watson. Metrics for security and performance in low-latency anonymity networks. In *Proceedings of Eighth International Symposium on Privacy Enhancing Technologies (PETS 2008)*, pages 115–132, Leuven, Belgium, July 2008. Springer.

[31] R. Naraine. Hacker builds tracking system to nab tor pedophiles. `http://www.zdnet.com/blog/security/hacker-builds-tracking-system-to-nab-tor-pedophiles/114`. [Online; accessed March 2011].

[32] Lasse Øverlier and Paul Syverson. Locating hidden servers. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*. IEEE CS, May 2006. `http://www.onion-router.net/Publications/locating-hidden-servers.pdf`.

[33] A. Pfitzmann and M. Waidner. Networks without user observability. *Computers and Security*, 6(2):158–166, April 1987.

[34] J. M. Schwier. *Pattern recognition for command and control data systems*. PhD thesis, Clemson University, August 2009.

[35] J. M. Schwier, R. R. Brooks, C. Griffin, and S. Bukkapatnam. Zero knowledge hidden markov model inference. *Pattern Recognition Letters*, 30(14):1273–1280, 2009.

[36] A. Serjantov and G. Danezis. Towards an information theoretic metric for anonymity. In P. Syverson and R. Dingledine, editors, *Proceedings of Privacy Enhancing Technologies, LNCS*, 2002. `citeseer.ist.psu.edu/serjantov02towards.html`.

[37] C. R. Shalizi. *Causal architecture, complexity, and self-organization in time series and cellular automata*. PhD thesis, University of Wisconsin-Madison, May 2001.

[38] C. R. Shalizi and J. P. Crutchfield. Computational mechanics: Pattern and prediction, structure and simplicity. *Statistical Physics*, 104(3-4):817–879, 2001.

[39] C. R. Shalizi and K. L. Shalizi. Blind construction of optimal nonlinear recursive predictors for discrete sequences. In M. Chickering and J. Y. Halpern, editors, *Artificial Intelligence: Proceedings of the Twentieth Conference*, pages 504–511, Arlington, VA, 2004. Uncertainty in Artificial Intelligence, AUAI Press.

[40] C. R. Shalizi, K. L. Shalizi, and J. P. Crutchfield. An algorithm for pattern discovery in time series. Technical Report arXiv:cs.LG/0210025, institution, November 2002. `http://arxiv.org/abs/cs.LG/0210025`.

[41] D. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on ssh. In *Proceedings of 10th USENIX Security Symposium*, pages 337–352, Washington, DC, August 2001.

[42] Y. Zhu, X. Fu, B. Graham, R. Bettati, and W. Zhao. Correlation-based traffic analysis attacks on anonymity networks. *IEEE Trans. Parallel Distribution Systems*, PP(99), August 2009.

[43] Y. Zhu, X. Fu, R. Bettati, and W. Zhao. Anonymity analysis of mix networks against flowcorrelation attacks. In *Proceedings of 2005 IEEE Global Telecommunications Conference (GLOBECOM 05)*, volume 3, pages 1801–1805, St. Louis, MO, 2005.